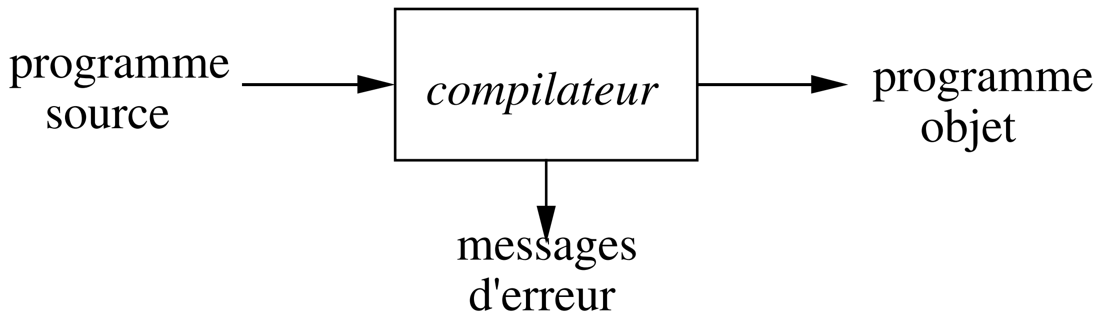
Introduction à la compilation

Un compilateur est un programme qui *traduit* un autre programme.



Premier compilateur: compilateur Fortran de J.Backus (1957)

Langage source: langage de haut niveau (Fortran, C, ...)

Langage cible: langage de bas niveau (assembleur, ...)

Cousins des compilateurs

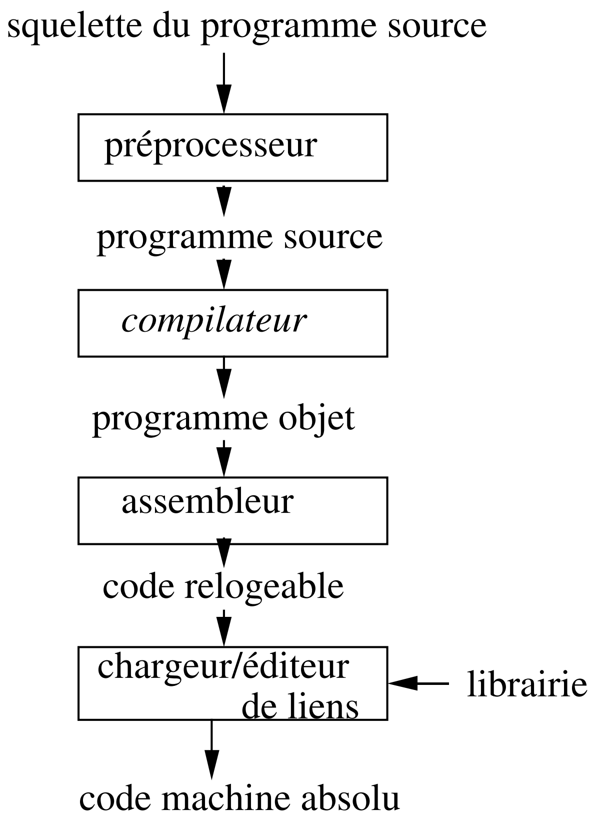
1. *Préprocesseurs*: effectue des substitutions de définitions, des transformations lexicales, des définitions de macros, etc.

2. *Interpréteurs*: diffère d'un compilateur par l'intégration de l'exécution et de la traduction. Utilisé pour les langages de commande.

3. *Formateurs de texte*: traduit la source dans le langage de commande d'une imprimante.

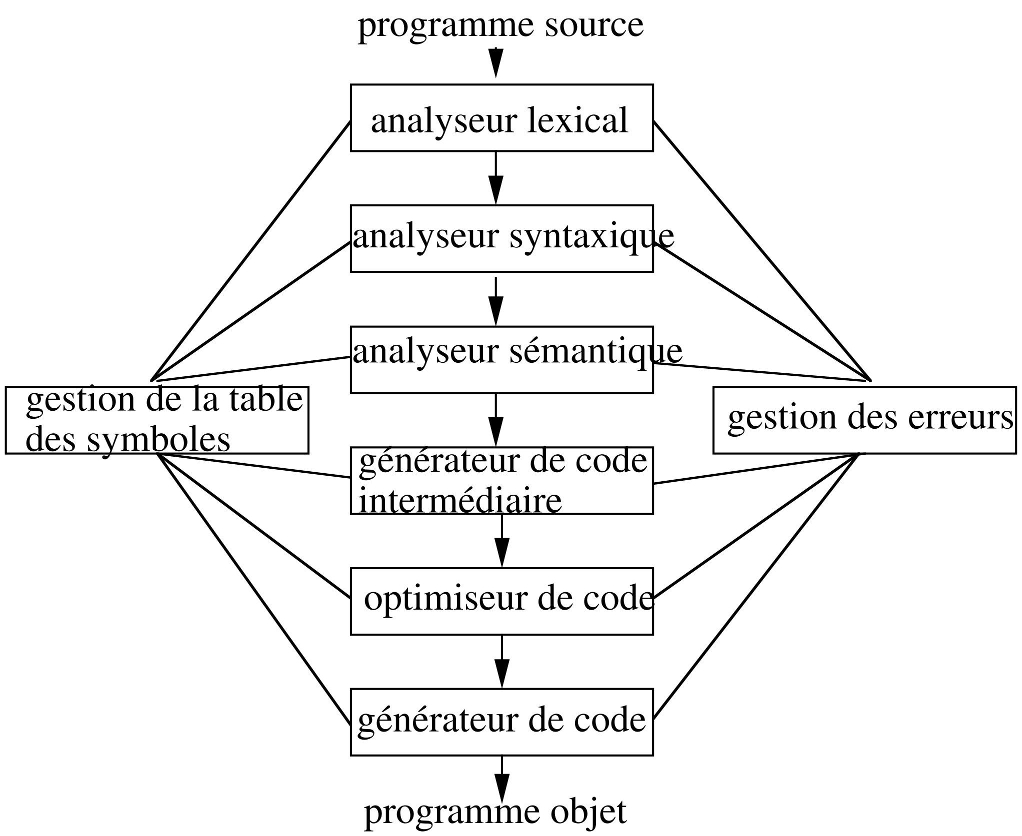
4. *Compilateurs de circuits*: traduit dans une description du circuit spécifié par la source.

L'environnement d'un compilateur



Liens avec l'éditeur de textes et le débugger.

Les phases de la compilation



Analyse lexicale

Analyse du programme source en constituants minimaux: les *lexèmes*.

On passe de

```
position := initial + rate*60
```

à

```
[id,1] [:=] [id,2] [+] [id,3] [*] [60]
```

Les identificateurs rencontrés sont placés dans la table des symboles.

Les blancs et les commentaires sont éliminés.

Analyse syntaxique

On reconstruit la structure syntaxique de la suite de lexèmes fournie par l'analyseur lexical.

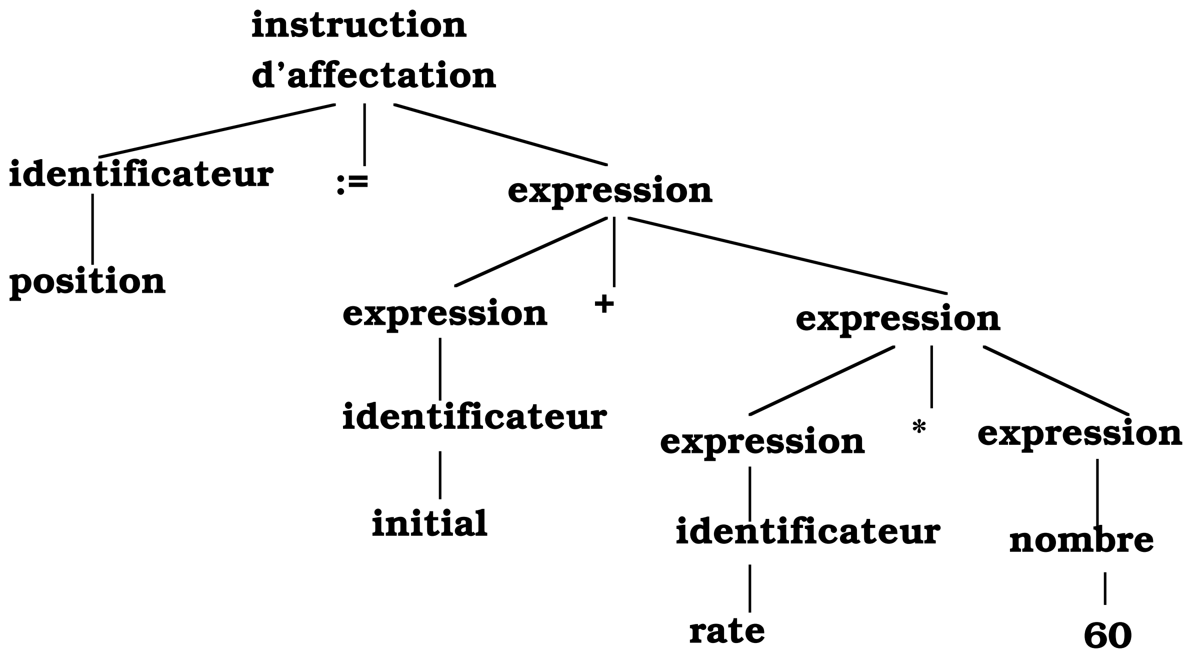


Figure 1: Arbre d'analyse

Génération de code intermédiaire

Programme pour une machine abstraite. Représentations utilisées:

1. Code à trois adresses
2. Arbres syntaxiques

Par exemple:

```
temp1 := intoreal(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
```

Optimisation du code

Elimination des opérations inutiles pour produire du code plus efficace. Par exemple, à partir du précédent:

```
temp1 := id3 *  60.0
id1 := id2 + temp1
```

1. La constante est traduite en flottant tout de suite
2. La variable `temp3` est éliminée.

Génération de code

La dernière phase produit du code machine relogeable. Un point important est l'utilisation des registres.

Exemple:

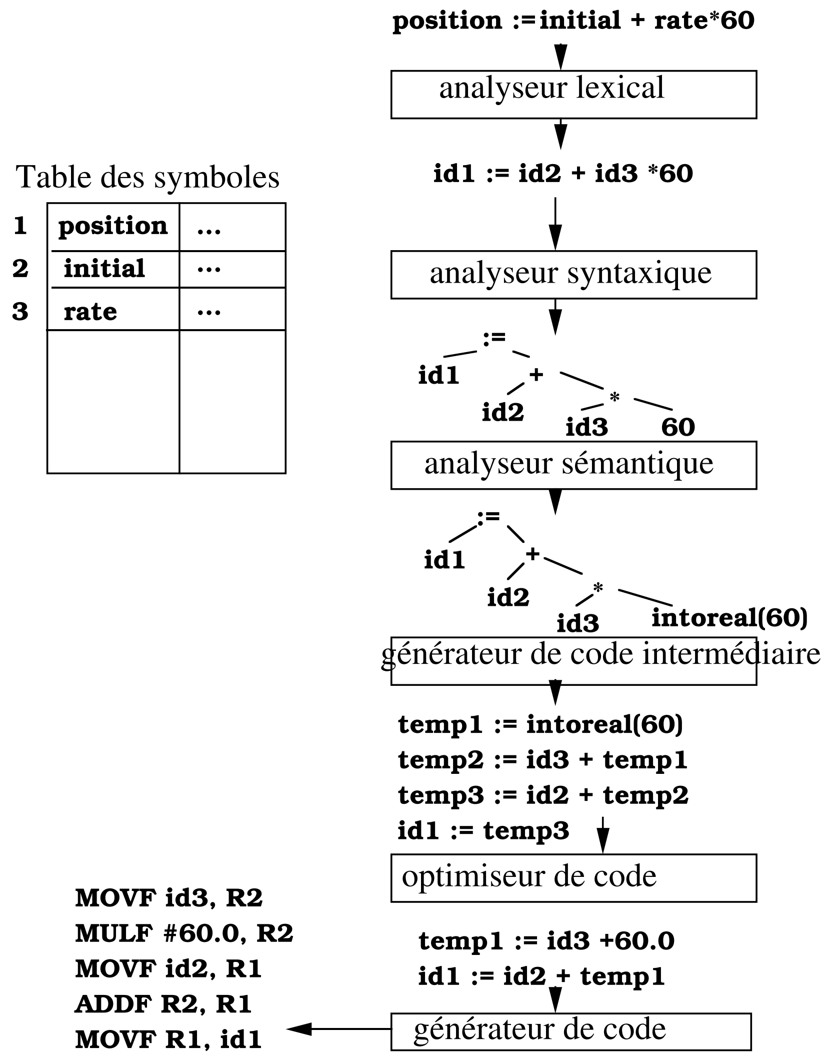
```
MOVF id3, R2
MULF #60.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1
```

Le F est pour flottant.

La première instruction transfère le contenu de `id3` dans le registre $R2$.

La seconde multiplie le contenu du regitre $R2$ par la constante 60.0.

...


position :=initial + rate*60
analyseur lexical
id1 := id2 + id3 *60
analyseur syntaxique
:=
id1
+
id2
*
id3
60
analyseur sémantique
:=
id1
+
id2
*
id3
intoreal(60)
générateur de code intermédiaire
temp1 := intoreal(60)
temp2 := id3 + temp1
temp3 := id2 + temp2
id1 := temp3
optimiseur de code
temp1 := id3 +60.0
id1 := id2 + temp1
générateur de code
MOVF id3, R2
MULF #60.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1
Table des symboles
1 position ...
2 initial ...
3 rate ...

Préprocesseurs

1. Traitement des macros
2. Inclusion de fichiers (commande `#include` en C)
3. Extensions de langages

Les définitions de macros permettent l'utilisation de paramètres formels.

Par exemple, en TEX, une définition de macro a la forme:

```
\define <nom> <mod\‘ele>{<corps>}
```

Par exemple, si on définit:

```
\define\JACM #1;#2;#3.
 {{\sl J. ACM} {\bf #1}:#2, pp. #3}
```

et si on écrit

```
\JACM 17;4;715-728
```

on doit voir

J. ACM **17**:4, pp. 715-728

Assembleurs

Les assembleurs sont des versions un peu plus lisibles du code machine avec des noms symboliques pour les opérations et pour les opérandes.

Par exemple:

```
MOV a, R1
ADD #2, R1
MOV R1, b
```

calcule `b := a+2`

Forme la plus simple: *assemblage en deux passes*. On crée une table des symboles (distincte de celle du compilateur) pour les identificateurs qui attribue une adresse mémoire. Par exemple:

IDENTIFICATEUR	ADRESSE
a	0
b	4

La seconde passe produit du code machine *relogeable*, i.e. qui peut commencer à n'importe quelle adresse L en mémoire.

Par exemple, les instructions précédentes peuvent être traduites en:

0001	01	00	00000000	*
0011	01	10	00000010	
0010	01	00	00000100	*

Les 4 premiers bits sont le code de l'instruction (avec 0001 pour load, 0010 pour store et 0011 pour add). Les deux bits suivants désignent un registre. Les deux bits suivants indiquent le mode d'adressage: 00 pour le mode adresse et 10 pour le mode littéral.

L'* désigne le '*relocation bit*' indiquant si l'adresse L doit être ajoutée ou non aux opérandes. Si $L = 00001111$, i.e. 15 alors a et b doivent être placés en 15 et 19 et le code absolu devient:

0001	01	00	00001111
0011	01	10	00000010
0010	01	00	00010011

Groupement des phases

1. *Parties frontale et arrière*: La partie frontale ('*front end*')regroupe tout ce qui ne dépend que du langage source et pas de la machine cible.

   La partie arrière ('*back end*') le reste. On peut utiliser la même partie frontale sur une machine différente.

2. *Passes*: Plusieurs phases peuvent être regroupées dans une même passe consistant en la lecture d'un fichier et l'écriture d'un autre.

   Par exemple: analyse lexicale, syntaxique, sémantique et génération de code intermédiaire peuvent être regroupés en une seule passe.

   Réduction du nombre de passes: technique du *recollement* ('*backpatching*').

Outils logiciels

Il existe un grand nombre de logiciels d'aide à la construction de compilateurs.

1. Générateurs d'analyseur lexical (*'scanner'*: construisent un *automate fini* à partir d'une spécification (expression rationnelle)

   Exemple: Lex sous Unix

2. Générateurs d'analyse syntaxique: génèrent un analyseur ('*parser*') à partir d'une grammaire;

   Exemple: Yacc sous Unix

3. Traducteurs: générent du code intermédiaire à partir d'un schéma de traduction:

   grammaire + règles sémantiques

Compilation d'un programme C

Si on compile par `gcc -S essai.c` le fichier

```
#include <stdio.h>

int main(void){
  printf("bonjour\n");
}
```

On obtient de l'assembleur 8086 avec

- `epb` pour *base pointer*
- `esp` pour *stack pointer*
- `pushl` pour *push long*
- etc...

```
       .file   "essai.c"
        .version        "01.01"
gcc2_compiled.:
                .section        .rodata
.LC0:
        .string "bonjour\n"
.text
        .align 16
.globl main
        .type    main,@function
main:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        subl    $12, %esp
        pushl   $.LC0
        call    printf
        addl    $16, %esp
        movl    %ebp, %esp
        popl    %ebp
        ret
.Lfe1:
        .size    main,.Lfe1-main
        .ident  "GCC: (GNU) 2.96 20000731 (Linux-Mandrake 8.0 2.96-0.48mdk)"
```

Compilation d'un programme Java

Si on compile par `javac Essai.java` le fichier

```
class Essai{

    public static void main(String[] args){
        System.out.println("bonjour");
    }
}
```

et qu'on exécute ensuite `javap -c Essai`, on obtient le bytecode

```
Compiled from Essai.java
class Essai extends java.lang.Object {
    Essai();
    public static void main(java.lang.String[]);
}

Method Essai()
   0 aload_0
   1 invokespecial #6 <Method java.lang.Object()>
   4 return

Method void main(java.lang.String[])
   0 getstatic #7 <Field java.io.PrintStream out>
```

```
3 ldc #1 <String "bonjour">
5 invokevirtual #8 <Method void println(java.lang.String)>
8 return
```