الكلية متعددة التخصصات - ورزازات
ⵜⴰⵖⵣⵍⵓⵜ ⵜⴰⵡⵜⵣⵎⵎⵉⵣⵜ – ⵍⵓⵕⵣⵓⵕⵜ
FACULTÉ POLYDISCIPLINAIRE DE OUARZAZATE
UNIVERSITÉ IBN ZOHR - AGADIR

Compilation

SMI/S5
Compilation
Prof. : M. BENADDY

Contenu du cours

- Introduction
- Analyse lexicale
  - Travaux pratiques (Lex,Flex)
- Analyse syntaxique et sémantique
  - Travaux pratiques (Yacc)
- Génération du code intermédiaire
- Optimisation du code

Déroulement

- 25 heures de cours
- 10 heures de TD
- 12 heures de TP

Evaluation

- Note de l'examen final
- Note des travaux pratiques et/ou mini projet

Références

- Compilers: Principles, Techniques, and Tools (2nd Edition), Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, Addison Wesley ; 2nd edition (2007)
- Modern Compiler Implementation in Java, 2nd ed. Andrew Appel, Cambrdige University Press (2004).
- Lex & yacc (2nd Edition 1992), John R. Levine, Tony Mason and Doug Brown, O'Reilly & Associates, Inc.
- Engineering a Compiler Second Edition, Keith D. Cooper Linda Torczon, Morgan Kaufmann, Elsevier, Inc. (2012).
- Programming Language, Third edition, Michael L. Scott, 2009 by Elsevier Inc.

الكلية متعددة التخصصات - ورزازات
ⵜⴰⵢⵙⴰⵍⵜ ⵜⴰⵅⵜⵙⵙⵉⵙⵜ – ⵡⵓⵔⵣⴰⵣⴰⵜ
FACULTÉ POLYDISCIPLINAIRE DE OUARZAZATE

Compilation

Chapitre 1 : Introduction générale

Introduction

- Au début de l'informatique on programme directement les ordinateurs en langage machine,
  - Tâche assez dur,
  - **Solution** : utiliser les possibilités de l'informatique pour faciliter le travail de programmation.
- Un ordinateur représente toutes les informations (données et programmes) par des suites de 0,1,
  - Le programmeur utilise des langages proches du langage humain !!
  - **Solution**: un traducteur <=> compilateur/ interpréteur

Langages compilés/interprétés

- **Langages compilés** : Se sont des langages où toutes les instructions sont traduites en code objet avant d'être exécutées cette conversion est effectuée par un ***compilateur***.
- **Langages interprétés** : Se sont les langages dont les instructions sont décodées et exécutées instruction/instruction lors de l'exécution du programme à l'aide d'un traducteur appelé ***interprète***.

Famille des langages

- Langages de bas niveau (1ère et 2ème génération) :
  - Le langage machine,
  - L'assembleur.
- Langages de haut niveau (3ème génération) :
  - les langages procéduraux (Fortran, Cobol, Basic, Pascal, Langage C, …)
  - les langages orientés objet (SIMULA, C++, Java, SMALTALK, Object Pascal, EIFFEL, …)
  - les langages orientés listes (LISP, PROLOG, PYTHON, ...)
- Langages de 4éme génération : se sont des langages créés pour des applications spécifiques (SQL, NOMAD, …)

Famille des langages

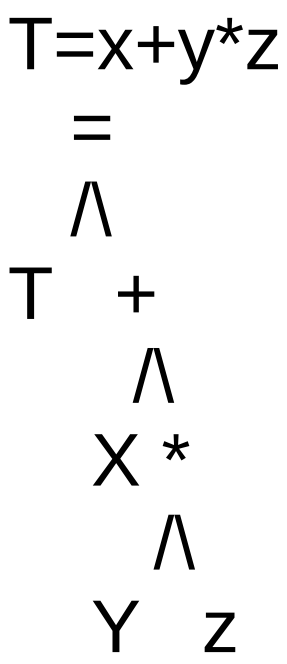
- **Remarque** : d'autres classifications peuvent être adoptés, telles que :
  - Les langages déclaratifs (C, C++, Java, C#...)
  - Les langages fonctionnels (ML, HASKELL,…)
  - Les langages de scripts (Awk, Javascript, Php, Perl, Python, Ruby, Tcl,...)

Les compilateurs

- Un compilateur est un programme qui lit le programme écrit dans un premier langage (langage source) et le traduit en un programme équivalent écrit dans un autre langage ‘’langage cible’’ ou langage destination.
- La compilation est composée de deux parties essentielles : l’analyse et la synthèse.
  - **L’analyse** : la partition du programme source en ces constituants et en créant une représentation intermédiaire.
  - **La synthèse** : génère le programme cible à partir de la représentation intermédiaire.
- **Remarque :** au cours du processus de traduction un rôle important du compilateur est de signaler à l’utilisateur la présence des erreurs dans le programme source

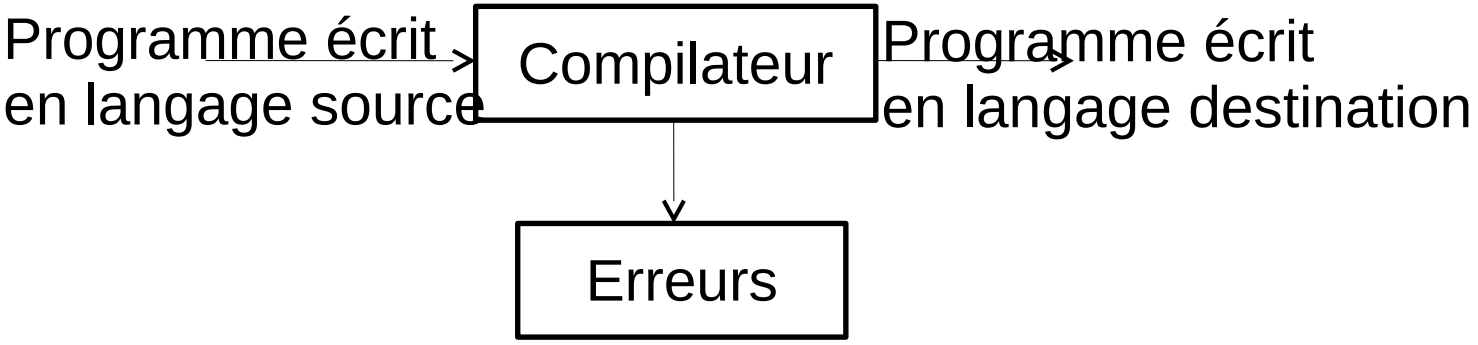
Les compilateurs

- **Remarque** : Pendant l'analyse les opérations de spécifiées par le programme source sont déterminées et conservées dans une structure hiérarchique appelée arbre abstrait chaque nœud de l'arbre représente une opération et les fils des nœuds représentent les arguments de cette opération.
  - **Exemple** : arbre abstrait



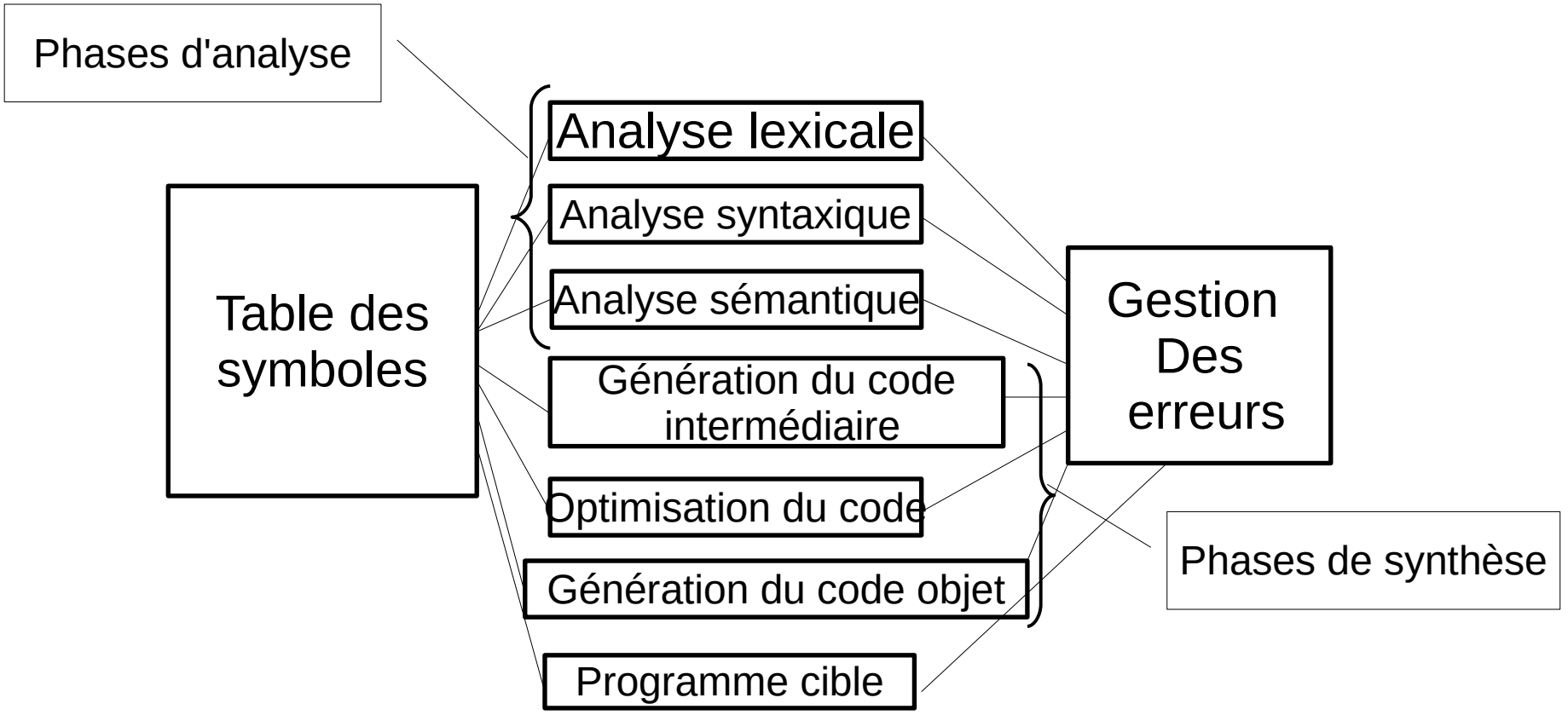
Les compilateurs

- **Représentation schématique** :



Les compilateurs

- **Phases d'un compilateur**



Les phases d'un compilateur

- **Analyse lexicale** (scanner) : Trouver les mots du programme.
  - Il décompose le texte du programme en lexème appelé aussi token ou unité lexicale, les tokens sont classés comme suit :
    - **Les identificateurs** : se sont les mots du programme formés par l'utilisateur pour l'appellation des variables et des constantes du programme,
    - **Les mots clés** : se sont les mots particuliers au langage ayant un sens bien spécifique, ils sont différents d'un langage à un autre,
    - **Les constantes,**
    - **Les opérateurs** : +, -,*,/, ou, et, non, …
    - **Les séparateurs** : se sont les caractères de ponctuation telle que le point virgule (;) la virgule (,), …
    - **L'affectation**.

Les phases d'un compilateur

- **Analyse lexicale**
  - **Exemple :**

Mots clé	Identificateurs	Constantes	Opérateurs	Séparateurs	Affectation
For, do, to	i, s	1, 10	+	;	:=

```
For i :=1 to 10 do
        s :=s+i ;
```

- L'analyseur lexicale remplit également tout les symboles du programme et qui n'appartient pas au langage comme les identifiants, les constantes, ..., dans la table des symboles.
- Le code produit par la phase lexicale est une chaîne d'unités lexicales,
- Une unité lexicale est codifiée par la paire (type, valeur (vallex)) où type est le type de l'unité lexicale à savoir les identificateurs, les constantes, ..., vallex est la valeur de l'unité : si l'unité est un identificateur vallex sera une chaîne de caractères qui forme le nom de cet identificateur.

Les phases d'un compilateur

- **Analyse lexicale**
  - **Exemple :** l'instruction position = initial + rate*60
    - Est codifié : <id,1> <=> <id,2> < +> <id,3>*<60>
  - **Remarque :** les mots clés ainsi que les séparateurs sont considérés comme des unités n’ayant pas de valeurs ils ne seront donc pas codifiés que par le type.

Les phases d'un compilateur

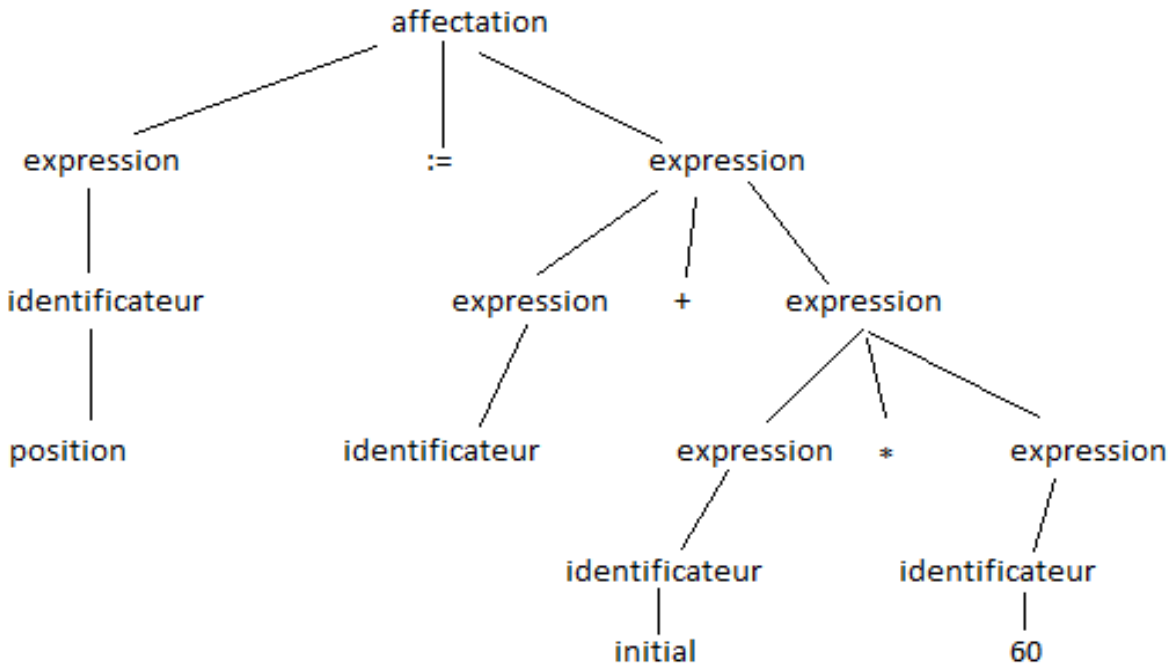
- **Analyse Syntaxique :** Appelée aussi analyse hiérarchique ou analyse grammaticale,
  - Vérifier si la syntaxe de la chaîne codée est conforme avec celle du langage à compiler,
  - Il consiste à trouver les phrases du programme source à partir des mots donnés par l'analyseur lexical, formellement cela consiste à créer l'arbre syntaxique, celui-ci décrit la structure du programme.
  - L'analyseur syntaxique se fait sur la base d'une grammaire. Généralement la structure hiérarchique d'un programme est exprimée par des règles récursives.
- **Exemple** :
  - Un identificateur est une expression,
  - Une constante est une expression,
  - Si expression 1 et expression 2 sont des expressions alors (expr1 + expr2) est une expression (expr1*expr2) l'est aussi.

Les phases d'un compilateur

- **Analyse Syntaxique :**
- **Exemple** : soit l'expression
  - position := initial + rate*60
  - position, initial, rate → expression
  - 60 → expression
  - Rate*60 → expression
  - initial + rate*60 → expression

Les phases d'un compilateur

- **Analyse Syntaxique :**
- **Exemple** : soit l'expression
  - position := initial + rate*60

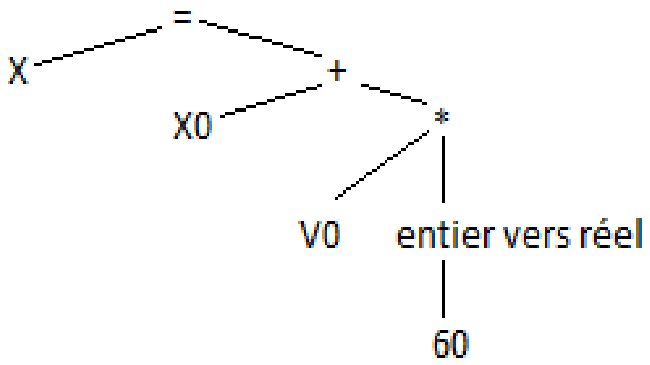


Les phases d'un compilateur

- **Analyse sémantique :** contrôle si le programme source contient des erreurs sémantiques elle utilise la structure hiérarchique déterminée par la phase d'analyse syntaxique pour identifier les opérateurs et les opérandes des expressions ainsi que les instructions.
  - Cette analyse effectuera quelques contrôles et quelques modifications de l'arbre syntaxique et contrôles portant par exemple :
    - Sur la visibilité des variables c-à-d vérifier qu'ils sont déclarées et visibles aux endroits où ils sont utilisés.
    - Sur le mode c-à-d vérifier les types utilisés. Ceci peut éventuellement amener des modifications de l'arbre syntaxique par l'ajout de nœuds spéciaux.
    - Par exemple pour le transtypage d'un entier vers un réel, on pourrait rajouter un nœud ''entier vers réel'' qui indique qu'une conversion doit être faite.

Les phases d'un compilateur

- **Analyse sémantique :**
  - **Exemple :** soit X = X0 + V0*60, sachant que X, X0, V0 sont des réels
  - L'arbre sémantique correspondant à l'arbre syntaxique est :

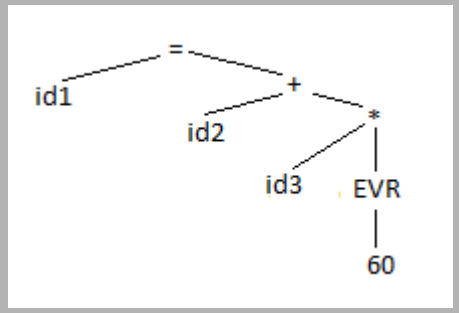


Les phases d'un compilateur

- **Production du code intermédiaire** : A partir des analyses syntaxiques et sémantiques un code simple de bas niveau (proche du code machine) est généré.
- Ce code intermédiaire doit :
  - Être facile à produire et facile à traduire en langage cible.
  - Cette représentation intermédiaire peut prendre une grande variété de forme, les plus utilisées sont :
    - **Le code post fixé** : c'est un code dans lequel tout opérateur apparaît après ses opérandes. C'est un code préconisé car les opérateurs et les opérandes dans ce code apparaissent dans le même ordre qu'à l'exécution.
    - **Le code à trois adresses** : c'est un code semblable au langage d'assemblage d'une machine. Il est caractérisé par le fait que l'instruction contienne en plus un seul opérateur et trois opérandes.

Les phases d'un compilateur

- **Production du code intermédiaire** :
- **Exemple** : id1 = id2 + id3*60




```
tmp1 = EVR(60)
tmp2 = tmp1*id3
tmp3 = tmp2+id2
id1 = tmp3
```

- Le compilateur doit décider dans quel ordre les opérations doivent être effectués, exemple : la multiplication précède l'addition.

Les phases d'un compilateur

- **Optimisation du code :** cette phase tante d'améliorer le code intermédiaire de façon que le code machine résultant s'exécute plus rapidement.
- L'optimisation peut se faire soit sur la taille du code soit sur la mémoire utilisée par le programme soit encore sur le temps d'exécution.
- **Par exemple** un algorithme produira le code intermédiaire en employant une instruction pour chaque opérateur de la représentation arborescente obtenue après l'analyse sémantique, bien qu'il y est une meilleure manière d'effectuer le même calcul avec les deux instructions :
  - tmp1 = id3 * 60
  - id1 = tmp1 + id2
- le compilateur peut déduire lui même que la conversion de l'entier 60 en réel peut être effectuée une fois pour toute au moment de la compilation de sorte que l'on peut éliminer l'opération entier vers réel. tmp3 n'est utilisée qu'une seule fois pour transmettre sa valeur à id1, on peut donc en toute sécurité substituer id1 à tmp3.
- Parmi les optimisations courantes :
  - Le code constant dans une boucle peut être sorti de celle-ci.

Les phases d'un compilateur

- **Optimisation du code :**
- Parmi les optimisations courantes :
  - Le code constant dans une boucle peut être sorti de celle-ci.

for(i = 0 ; i<3 ; i++)
{
scanf(‘’%d’’,&a) ;
b = 12 ;
a = a + i ;
printf(‘’%d’’,a) ;
}
	Opt	b = 12 ;
		for(i = 0 ; i<3 ; i++)
		{
		scanf(‘’%d’’,&a) ;
		a = a + i ;
		printf(‘’%d’’,a) ;
		}

  - Les calculs et les transtypages portant uniquement sur des constantes peuvent être faites à la compilation.
  - Les parties du code inaccessibles peuvent être supprimées.

If(cond1 || true) Inst1 ; else inst2 ;	Opt	Inst1 ;

Les phases d'un compilateur

- **Génération du code :** cette phase produit le code final équivalent au programme de départ.
- Avant la génération du code il faut connaître la forme du code final et aussi l'architecture de la machine sur laquelle le programme sera exécuté.
- Généralement le code cible consiste en langage d'assemblage.
- On sélectionne les emplacements mémoire pour chacune des variables utilisées dans le programme, ensuite les instructions intermédiaires sont traduites en une suite d'instructions machines qui effectuent la même tâche,

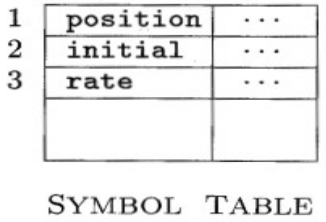
Les phases d'un compilateur

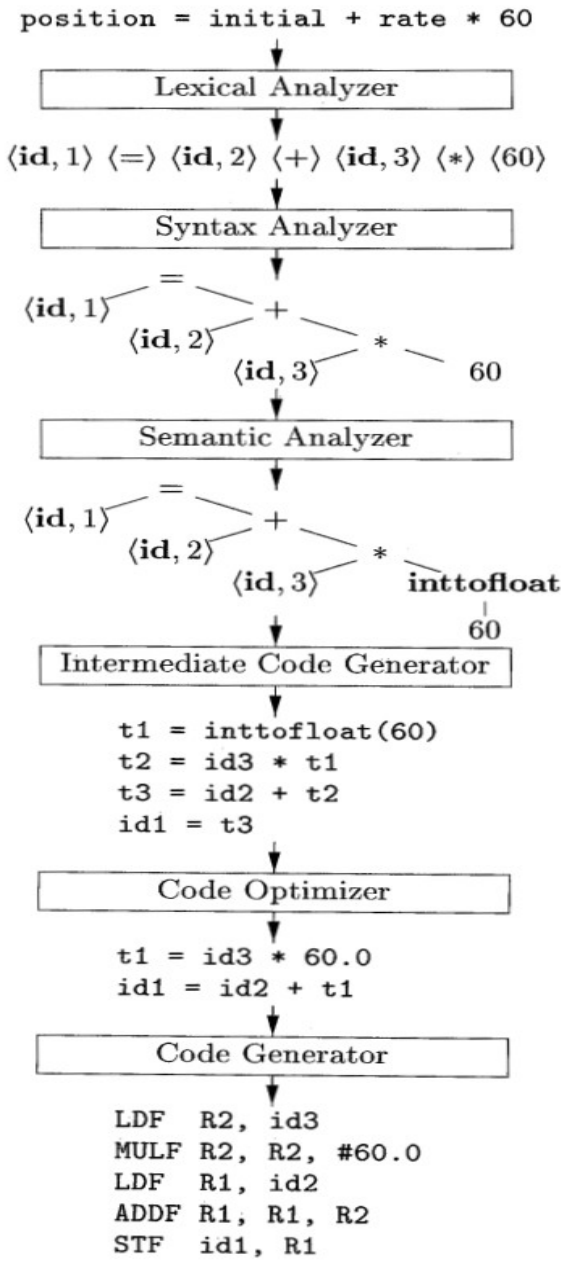
- **Optimisation du code :**
- Un aspect crucial de ce processus est l'assignation de variables aux registres, par exemple en utilisant les registres R1 et R2 la traduction de l'instruction id1 = id2 + id3*60 est :

MOVF id3,R2
MULF #60.0,R2
MOVF id2,R1
ADDF R2,R1
MOVF R1,id1
	Le F signifie float (réel)
	# constante interne

Les phases d'un compilateur

- **Synthèse** :





Les phases d'un compilateur

- **Gestion de la table des symboles** :
  - Toutes les phases de compilation communiquent des informations entre elles par l'intermédiaire de la table des symboles,
  - Chaque phase installe dans cette table les informations nécessaires à l'accomplissement des phases ultérieures.
    1) L'analyse lexicale insère les identificateurs, les constantes dans la table des symboles.
    2) L'analyse syntaxique est sémantique associe à chaque identificateur sont type, à chaque fonction son nombre d'argument et ses arguments, etc.

Les phases d'un compilateur

- **Gestion de la table des symboles** :

  3) Le générateur du code intermédiaire doit accéder à la table pour repérer le types des opérandes afin de vérifier la compatibilité des types. Si le langage permet le mixage des types, une instruction spécifiant une conversion de type sera générée. Le générateur doit mémoriser le nouveau type de l'identificateur dans la table des symboles.

  4) Lors de l'optimisation du code à tout identificateur résultant par exemple une expression à éliminer doit être substituée par l'identificateur de l'expression à garder, le résultat de cette modification doit figurer dans la table.

  5) On doit insérer dans la table des symboles l'information d est remplacée par a dans la suite du code.

  6) Dans la génération du code final on peut associer à chaque identificateur les registres qui doivent contenir sa valeur.

B = 12 ; C = 14 ; A = B + C ; D = B + C ;	opt	B = 12 ; C = 14 ; A = B + C ; (id,d)

Les phases d'un compilateur

- **Gestion des erreurs** : La gestion des erreurs est une étape importante du compilateur, il existe deux stratégies :
  1) Soit on arrête dés qu'il y a une erreur.
  2) Soit on essaie de resynchroniser et de déterminer le plus précisément possible la cause de l'erreur.
- Une bonne gestion des erreurs permet de détecter un maximum en même temps, ce qui permet de les corriger avant de réessayer la compilation.
- L'inconvénient de cette stratégie est qu'on peut avoir d'autres messages d'erreurs dues à des erreurs précédentes.

Les phases d'un compilateur

- **Gestion des erreurs** :
- Chaque phase de compilation détecte son propre type d'erreur :
  - L'analyse lexicale détecte peu d'erreurs essentiellement les fautes de frappe, il s'agit de caractères interdits ou ayant une forme illégale pour le langage (3x, β, ф, ...).
  - L'analyse syntaxique détecte les erreurs de construction du programme par exemple : en C une accolade rencontre un point manquant ou encore fi(x==4) (fi est considéré comme identificateur par l'analyseur lexical, mais l'analyseur syntaxique dit que cette instruction est illégale.
  - L'analyse sémantique détecte des erreurs de type, les variables non déclarées, etc.
  - L'optimiseur du code peut informer une instruction du programme ne peut jamais être accessible et par conséquent jamais exécutée.
  - Le générateur du code objet peut indiquer qu'une valeur créée par le compilateur est trop grande pour être mémorisée.

Les phases d'un compilateur

- **Regroupement des phases :**
  - Les sections précédentes traite l’organigramme logique d’un compilateur, pour accélérer le processus de compilation, les différentes phases sont regroupées en une phase.

Les phases d'un compilateur

- **Regroupement des phases :**
- **Parties frontale et finale :**
  - Les différentes phases sont souvent remise en une partie frontale et une partie finale.
    - La partie frontale est constituées des phases qui dépendent principalement du langage source, elle comprend l'analyse lexicale, l'analyse syntaxique, la création de la table des symboles, l'analyse sémantique, la production du code intermédiaire et le traitement des erreurs associes à chacune de ces phases.
    - La partie finale comprend les portions du compilateur qui dépendent de la machine cible, on trouve dans cette partie certains aspects, de l'optimisation du code ainsi que la production du code objet avec les opérateurs nécessaires de traitement des erreurs et de gestion de la table des symboles.

Les phases d'un compilateur

- **Regroupement des phases :**
- **Les passes :**
- Une passe est un module qui rassemble plusieurs phases du compilateur d'une exécution.
- Dans une exécution les différentes phases chevauchent entre elles (en relation) que leurs activités sont entrelacées durant l'exécution de cette passe.
- Il existe un choix entre deux options d'implantation.
  1) La première consiste à l'utilisation d'un sous programme pour chaque phase du processus, appelé aussi le parallélisme.
  2) La 2ème consiste à donner le contrôle d'une phase qui fait appelle au module des autres phases au moment opportun.

Les phases d'un compilateur

- **Outils de construction de compilateurs :**
  - Les grammaires hors contextes, pour l’analyse syntaxique, et les expressions régulières, pour l'analyse lexicale sont deux abstractions les plus utiles utilisés dans compilateurs modernes.
  - Lex/Flex (qui converti des déclarations en un programme d’analyse lexicale).
  - Yacc (Yet Another Compiler-Compiler), qui convertit une grammaire en un programme d’analyse syntaxique.