Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Informatique

Master 1 : IV

Année Universitaire : 2024/2025
Module Compilation

Projet de compilation du langage PHYLOG
Avec les Outils FLEX et BISON

1. Introduction :

Le but de ce projet est de réaliser un mini-compilateur en effectuant les différentes phases de la compilation à savoir l'analyse lexicale en utilisant l'outil FLEX et l'analyse syntaxico-sémantique en utilisant l'outil BISON, du langage **« PHYLOG »**.

Les traitements parallèles concernant la gestion de la table des symboles ainsi que le traitement des différentes erreurs doivent être également réalisés lors des différentes phases d'analyse du processus de compilation.

2. Description du Langage PHYLOG:

La structure générale d'un programme est la suivante :

nom_du_programme
DATA
……
END
CODE
……
END
END

- ***Commentaire :***

Un commentaire est ignoré par le compilateur. Exp : **§** commentaire

2.1. **Déclarations :**

Le bloc déclaration comporte des variables et des constantes.

2.1.1. Déclaration des variables de type simple:

La déclaration d'une variable a la forme suivante :

TYPE: *list_variables ;*

- ***liste_variables***

Ensemble d'identificateurs séparés par une bare.

Exemple :

§Déclaration d'une variable simple

type: *nom_variable1* **;**

§Déclaration de 2 variables

type: *nom_variable2* | *nom_variable3 ;*

2.1.2. déclaration des tableaux

La déclaration d'un tableau a la forme suivante :

VECTOR : nom [borne_inf, taille : type] ;

2.1.3. Type :

Le type peut être : INTEGER, CHAR, STRING, FLOAT.

- **INTEGER :** Une constante entière est une suite de chiffres. Elle peut être signée ou non signée tel que sa valeur est entre **-32768** et **32767**. Si la constante entière est signée, elle doit être mise entre parenthèses.
- **FLOAT :** Une constante réelle est une suite de chiffres contenant le point décimal. Elle peut être signée ou non signée. Si la constante réelle est signée, elle doit être mise entre parenthèses.
- **CHAR :** Une variable de type CHAR représente un caractère.
- **STRING :** Une variable de type STRING représente une chaîne de caractères.

2.1.4. Déclaration des Constantes:

La déclaration d'une constante se fait comme suit :

CONST: *idf = valeur ;*

- **CONST :** une constante peut prendre une valeur unique correspondant à un des types cités précédemment et qui resté inchangée tout au long du programme.

	Exemple
INTEGER	**Declaration: INTEGER:** a ; **Valeurs de a possible:** 21, (-6)
FLOAT	**Declaration: FLOAT:** a ; **Valeurs de a possible:** 88.5,(-0.24), (+3.0)
CHAR	**Declaration: CHAR:** a ; **Valeurs de a possible:** 'a', ' !'
STR	**Declaration: STRING:** a ; **Valeurs de a possible:** "chaine de caractères"
CONST	**Declaration: CONST:** a =valeur ; **Valeurs de a possible:** "chaine de caractères", 'c',5, (-6),5.6

2.1.5. Identificateur :

Un identificateur est une suite alpha numérique qui commence par une lettre majuscule suivie d'une suite de chiffres et lettres minuscules. Un IDF ne doit pas contenir plus de 8 caractères.

Les noms du programme principal et des variables et des constantes sont des identificateurs.

Opérateurs arithmétique, logique et de comparaison

Opérateurs arithmétique : +, - , *, /

Opérateurs logique
(expression1. AND. expression2) : le et logique.
(expression1. OR . expression2) : le ou logique.
(NOT expression) : la négation.

Opérateurs de comparaison

(expression1.GE. expression2) : expression1 > expression2.
(expression1.L. expression2) : expression 1< expression2.

(expression1.GE. expression2) : expression1 ≥ expression2.
(expression1.LE. expression2) : expression1 ≤ expression2.
(expression1.EQ. expression2) : expression1 = expression2.
(expression1.DI. expression2) : expression1 ≠ expression2.

Les conditions

Une condition est une expression qui renvoie une valeur booléenne. Elle peut prendre la forme d'une expression logique, de comparaison ou une valeur booléenne.

2.2. **Instructions :**

Affectation	
Description	Exemple
Idf = expression ;	A=(X+7+B)/ (5,3-(-2)) ; A= 'c'**;** A="hello"**;**

Entrées / Sorties	
Description	Exemple
Entrée : **READ (" signe de formatage": @ idf);** Sortie : **DISPLAY**(" voilà la valeur de idf ***signe de formatiage***" : idf**) ;**	**READ ("$":@ idf_entier);** **READ ("%":@ idf_float);** **READ ("#":@ idf_string);** **READ ("&":@ idf_char);** **DISPLAY ("c'est un entire $": idf_entier);** **DISPLAY ("c'est un reel %":idf_float);** **DISPLAY ("c'est une chaine de caractères #":idf_string);** **DISPLAY ("c'est un caractère&":idf_char);**

Condition IF (Si … Alors … Sinon … Fin Si**)**	
Description	Exemple
IF(condition): instruction 1 instruction2 **ELSE :** instruction 3 instruction4 **END** **Note :** Le premier bloc est exécuté ssi la condition est vérifiée. Sinon le bloc « ELSE » sera exécuté s'il existe. On peut avoir des conditions imbriquées.	IF (Aa.GE.Bb): Cc=E+2.6; ELSE: DISPLAY ("la valeur de A est $": A); END

Boucle (Faire si … faire fait**)**	
Description	Exemple
FOR (idf: PAS: Condition_d'arrêt) instruction 1 instruction2 **END** **Note :** le bloc d'instructions est exécuté ssi la condition est vérifiée. On peut avoir des boucles imbriquées.	FOR (i :2 :n) READ (" %": A); END

Remarque :

1- **Toute instruction doit terminer par un point-virgule.**
2- **On ne peut lire qu'une seule variable à la fois.**
3- **L'instruction « READ » ne peut contenir que le format de la variable à lire.**
4- **Une chaine de caractère est une suite de caractères situés entre deux guillemets.**
5- **Un idf de type CHAR est un seul caractère situé entre deux apostrophes.**

- ***Associativité et priorité des opérateurs :***

Les associativités et les priorités des opérateurs sont données par la table suivante par ordre croissant :

Opérateur		**Associativité**
Opérateurs Logiques	OR (ou)	Gauche
	AND (et)	Gauche
Opérateurs de comparaison	G (>) GE (>=) EQ (==) DI (!=) LE(<=) L(<)	Gauche
Opérateurs Arithmétiques	+ -	Gauche
	* /	Gauche

3. Analyse Lexicale avec l'outil FLEX :

Son but est d'associer à chaque mot du programme source la catégorie lexicale à laquelle il appartient. Pour cela, il est demandé de définir les différentes entités lexicales à l'aide d'expressions régulières et de générer le programme FLEX correspondant.

4. Analyse syntaxico-sémantique avec l'outil BISON :

Pour implémenter l'analyseur syntaxico-sémantique, il va falloir écrire la grammaire qui génère le langage défini au-dessus. La grammaire associée doit être LALR. En effet l'outil BISON est un analyseur ascendant qui opère sur des grammaires LALR. Il faudra spécifier dans le fichier BISON les différentes règles de la grammaire ainsi que les règles de priorités pour les opérateurs afin de résoudre les conflits. Les routines sémantiques doivent être associées aux règles dans le fichier BISON.

6. Gestion de la table de symboles :

La table de symboles doit être créée lors de la phase de l'analyse lexicale. Elle doit regrouper l'ensemble des variables et constantes définies par le programmeur avec toutes les informations nécessaires pour le processus de compilation. Cette table sera mises à jour au fur et à mesure de l'avancement de la compilation. Il est demandé de prévoir des procédures pour permettre de **recherche** et d'**insérer** des éléments dans la table des symboles (table de hachage). Les variables structurées de type tableau doivent aussi figurer dans la table de symboles.

7. Génération du code intermédiaire

Le code intermédiaire doit être générer sous forme des quadruplets.

8. Optimisation

9. Le code assembleur

10. Traitement des erreurs :

Il est demandé d'afficher les messages d'erreurs adéquats à chaque étape du processus de compilation. Ainsi, lorsqu'une erreur lexicale ou syntaxique est détectée par votre compilateur, elle doit être signalée le plus précisément possible, par sa nature et sa localisation dans le fichier source. On adoptera le format suivant pour cette signalisation :

Type_ de_ l'erreur, line 4, colonne 56: entité qui a générée l'erreur.