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

L3 ACAD- Section A, B et C
Année Universitaire : 2019/2020 **Module : Compilation**

Projet de compilation du langage Mini-ACAD 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 «**Mini-ACAD**».

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 «Mini-ACAD»

La structure générale d'un programme est la suivante:

```
PROGRAM nom_du_programme
 //  Déclaration des variables et des constantes//
BEGIN
     ……..
END
```

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:

var liste_variables**: TYPE £**

- liste_variables: représente l'ensemble d'identificateurs séparés par deux pipes ||

Exemple:

//Déclaration d'une variable simple//

var *nom_variable :* **type £**

//Déclaration de 2 variables//

var *nom_variable1* || *nom_variable2:* **type £**

2.1.2 Déclaration des tableaux

La déclaration d'un tableau a la forme suivante:

var nom_tableau [taille]**: [TYPE] £**

- ***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.

2.1.3 Déclaration des Constantes:

La déclaration d'une constante se fait comme suit:

let *variable****: TYPE=*** *valeur £*

- let: 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	**Let a INT = 15** £
FLOAT	**Let a FLOAT= 3.14 £**
CHAR	**Let a CHAR='r' £**
STRING	**Let a STRING: "*abc*"*£***
CONST	**Let a** =valeur £

2.2 Les commentaires

Un commentaire peut être écrit sur une ou sur plusieurs lignes en mettant le texte entre « // » et « // »

2.3 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.

2.4 Opérateurs arithmétique, logique et de comparaison

Opérateurs arithmétique: +, - , *, /
Opérateurs logiques
(expression1 >> expression2): expression1 > expression2.
(expression1 << expression2): expression 1< expression2.
(expression1 >>= expression2): expression1 ≥ expression2.
(expression1 <<= expression2): expression1 ≤ expression2.
(expression1 == expression2): expression1 = expression2.
(expression1<<>> expression2): expression1 ≠ expression2.

2.5 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.

2.6 Les instructions:

<table>
<tr><td colspan="3">Affectation</td></tr>
<tr><td>Description</td><td>Exemple</td><td></td></tr>
<tr><td>Idf = expression £</td><td>A=(X+7+Z)/ (15,6-(-4)) £
A= 'c' £
A="Good bye" £</td><td></td></tr>
<tr><td colspan="3">Entrées / Sorties</td></tr>
<tr><td>Description</td><td colspan="2">Exemple</td></tr>
<tr><td>Entrée :
GET (" signe de formatage": @ idf)£

Sortie :
SHOW(" voilà la valeur de idf signe de formatiage" : idf) £</td><td colspan="2">GET("$":@ idf_entier)£
GET("%":@ idf_float)£
GET("#":@ idf_string)£
GET("&":@ idf_char)£

SHOW("Variable entière $": idf_entier)£
SHOW("Variable réelle %":idf_float)£
SHOW("Une chaîne de caractères #":idf_string)£
SHOW("Un caractère&":idf_char)£</td></tr>
</table>

Condition IF (Si … Alors … Sinon … Fin Si**)**	
Description	Exemple

IF(condition):
{ instruction 1
instruction2
RETURN (valeur/ instructionN)
}
ELSE :
{ instruction 3
instruction4
RETURN (valeur/ instructionN)
}
 	
Note:
Le premier bloc est exécuté si et seulement si la condition est vérifiée. Sinon le bloc « ELSE » sera exécuté s'il existe.
On peut avoir des conditions imbriquées.
	IF (Aa >> Bb)
	{
	Cc=Ef+2.6 £
	RETURN (Cc) £
	}
	ELSE:
	{
	RETURN(Cc)
	}
	ENDIF

Boucle (Faire si … faire fait**)**	
Description	Exemple
FOR (idf: PAS: Condition_d'arrêt) instruction 1 instruction2 **END_FOR** **Note :** le bloc d'instructions est exécuté ssi la condition est vérifiée. On peut avoir des boucles imbriquées.	**FOR** (x :2 : ie<<N) GET($":@ x)£ **END_FOR**

Remarques:

1. Toute instruction doit terminer par le symbole £.
2. On ne peut lire qu'une seule variable à la fois.
3. Une chaîne de caractère est une suite de caractères situés entre deux **guillemets**.
4. 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 de comparaison	>> (>) >>= (≥) == (==) <<>> (!=) <<= (<=) << (<)	Gauche
Opérateurs Arithmétiques	+ -	Gauche
	* /	Gauche

1. 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.

2. 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 générateur d' analyseurs ascendants 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.

3. 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 mise à jour au fur et à mesure de l'avancement de la compilation. Il est demandé de prévoir des procédures pour permettre de **rechercher** et d'**insérer** des éléments dans la table des symboles. Les variables structurées de type tableau doivent aussi figurer dans la table de symboles.

4. 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, ligne 4: entité qui a générée l'erreur.