Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Electronique et d'Informatique
Département d'Informatique
Master IL et IV, Année Universitaire : 2020/2021, Module Compilation

1-Introduction

Le but de ce projet est de réaliser un mini-compilateur du langage décrit ci-dessous, en effectuant les différentes phases de la compilation : lexicale, syntaxique, sémantique et génération du code intermédiaire.

La structure générale

CODE nom-program
    <liste-déclaration>
START
    <liste-instruction>
END.

Les caractéristiques du langage

- Un idenficateur (idf) est une suite alphanumérique commençant par une lettre **Majuscule** et ne dépassant pas 20 caractères.
- Une constante entière est une suite de chiffres dont la longueur ne dépasse pas 6 caractères. Elle peut être signée ou pas.
- Une consante réelle est une suite de chiffres contenant le point décimal. Elle peut être signée ou pas. Sa taille maximale est de 8 caractères (signe et . compris).
- Le programme peut contenir un ou plusieurs commentaires. Un commentaire est une suite de caractères quelconques compris entre $ et $, et ne contient pas $.
- nom-program est un identificateur.
- Tout programme se termine par un END suivi d'un point.
- Les motes clés sont écrits en majuscules.

La partie déclaration

On déclare les identifacteurs ainsi que leur type sous la forme :

<Type> liste-idf;

- Liste-idf peut être composée d'un seul idf ou bien d'une liste d'idfs séparés par une virgule.
- Il y a 4 types : **INTEGER, REAL, CHAR et STRING.**
- On doit déclarer aussi toute constante utilisée comme suit : **CONST** idf=constante;

La partie instruction

Dans cette partie, chaque instruction se termine par un point-virgule. On peut avoir les instructions suivantes :

- **Affectation**
- **Boucle**
- **Contrôle**

Instruction	Description	Exemples
Affectation	idf:=expression arithmétique; Ou bien Idf:=constante; Idf:=**PROD**(exp1,..,expn); Sémantique: L'identificateur Id reçoit le produit des expressions Strictement positives parmi les n expressions données Entre parenthèses. Si aucun expression n'est positive, Idf reçoit 1. n>1.	A:=2; A:=C+D; A:=-3; Idf:=PROD(A+B, C*D);

Boucle	**WHILE** condition **EXECUTE**
	{
	Instructions
	};
	Condition : c'est une expression logique.
	Exemple :
	I GT J
	I LT 3
	I+5 EQ J
		WHILE I LT J
		EXECUTE
		{ A:=2;
		A:=C+D/8;
		};

Contrôle	**WHEN** condition **DO** instruction; **OTHERWISE** Instruction;	WHEN J EQ I DO A:=C+D; OTHERWISE I:=5;

- Les expressions arithmétiques sont composées des opérateurs : +, -, *, / .
- Opérateurs de comparaison sont : EQ, LT, GT, LE, GE, NE.

Associativité et priorité des opérateurs

- **Associativité** : gauche pour les opérateurs.
- **Priorité** : les priorités sont données par la table suivante :

	Opérateur	Associatvité
Opérateurs de Comparaison	GT (>) GE (>=) EQ (=) NE (!=) LE (<=) LT (<)	Gauche
Opérateurs Aritmétiques	+ -	Gauche
	* /	Gauche

Travail à réaliser :

Ci-dessous les différentes phases à effectuer afin de réaliser le compilateur demandé.

- **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.

- **Analyse syntaxique avec l'outil BISON**
  Pour implementer l'analyseur syntaxique, 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 priorité pour les opérateurs afin de résoudre les conflits. Les routines sémantiques doivent être associées aux règles dans le fichier BISON.

- **Analyse sémantique :** Parmi les erreurs à identifier, nous pouvons citer :
- Idf non déclaré
- Idf double déclaration
- Non compatibilité de type
- Division par zéro

- **Gestion de la table des symboles**
  La table des symboles doit être crée lors de la phase de l'analyse lexicale. Elle doit regrouper l'ensemble des identificateurs et constantes définis 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.
  La table doit avoir au minimum les champs suivantes :
  - Nom : qui indique le nom de l'identificateur ou constante.
  -Type : type de l'identificateur ou de la constante.

- **Génération de code intermédiaire** sous forme de quadruplets, et ceci pour toutes les instructions possibles.

- **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 généré l'erreur.