Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Informatique - Département IA et SIQ

Licence 3 : ISIL B Année Universitaire : 2024/2025

Projet de réalisation d'un mini compilateur pour un mini langage avec les Outils FLEX et BISON

1. Introduction

Le projet vise à développer un mini-compilateur pour un langage de programmation simplifié nommé EasyCode. Ce langage dispose de ses propres règles de syntaxe et de structures, et prend en charge des concepts fondamentaux comme les variables, les types de base, les boucles, et les conditions. Le projet inclut l'intégration des différentes phases essentielles de la compilation : l'analyse lexicale, syntaxique et sémantique, ainsi que la génération de code intermédiaire.

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

2. Syntaxe Générale

Structure du Programme :

- Le programme commence par `DEBUT` et se termine par `FIN`.
- Les blocs de code sont délimités par `{` et `}`.

Structure générale du programme :

```
DEBUT
        # Déclaration des variables
    <Déclaration des variables>
        # Partie Instructions
EXECUTION
{
```

```
        <instructions>
 }
FIN
```

3. Détails des Caractéristiques de EasyCode

3.1. Commentaires

Les commentaires dans EasyCode utilisent des symboles de délimitation uniques :

- **Commentaire sur une seule ligne** : commence par `--` et se termine avec un symbole de fin `##`.
    - Exemple :

```
-- Ceci est un commentaire ##
```

- **Commentaire multi-lignes** : commence par `{-` et se termine par `-}`.
    - Exemple :

```
{-
Ce commentaire
couvre plusieurs lignes.
-}
```

3.2. Déclarations de Variables

EasyCode supporte les types suivants :

Types de base : `NUM` (entier), `REAL` (réel), `TEXT` (chaîne de caractères).

- **NUM** : Représente les entiers signé ou non. Gamme : -32768 à 32767.

Exemple de valeurs : `25`, `-15`, `0`

- **REAL** : Représente les réels (nombres à virgule flottante). Peut être signé ou non.

Exemple de valeurs : `3.14`, `-0.5`, `2.0`

- **TEXT** : Représente les chaînes de caractères

Exemple de valeurs : `"Bonjour"`, `"Nom"`

Déclaration de Tableaux

- **Syntaxe** : `<type> : <nom_tableau>[taille];`
- **Exemple** : `NUM : scores[10];`
- **Notes** :
  - La taille doit être un nombre entier positif.
  - L'accès aux éléments du tableau se fait par index (`scores[0]`, `scores[1]`, etc.).

- **Syntaxe de Déclaration** : `<type> : <nom_variable>;`
  - Exemple :

```
NUM : age;
REAL : pi;
TEXT : nom;
```

Identificateurs:

- Une suite alphanumérique (lettres et chiffres) débutant par une lettre majuscule, elle peut contenir un underscore (_).
- Longueur maximale : 10 caractères.
- Le nom du programme principal, les variables, les constantes et les structures sont des identificateurs.

- **Exemples : `total`, `compteur`, `valeur_1`**
- **Déclaration de constantes** : utilise le mot-clé `FIXE` pour indiquer une constante.
  - Syntaxe :

```
FIXE <TYPE> : <nom_variable> = <valeur>;
```

  - Exemple :

```
FIXE NUM : MAX_VALEUR = 100;
```

3.3. Instructions

Affectation

Syntaxe	Exemple
<Identificateurs> <- <Expression>	A <- 2 ; A <- H+D ; Tab[1] <- a+b/4 ;

Condition (SI - ALORS - SINON)

Syntaxe	Exemple

SI (<condition>) ALORS {
<instructions si vrai>
}
SINON {
<instructions si faux>
}
	SI (age > 18) ALORS {
	affiche("Majeur")
	} SINON {
	affiche("Mineur")
	}

Boucle TANTQUE

Syntaxe	Exemple

TANTQUE <condition> FAIRE {
<instructions>
}
	TANTQUE (compteur < 10) FAIRE
	{
	compteur = compteur + 1
	}

Remarque : on peut avoir des instructions imbriquées.

4. Opérations et Expressions

Opérations et Priorités

4.1 Opérateurs Primitifs

- **Arithmétiques** :
  - `+` : addition
  - `-` : soustraction
  - `*` : multiplication
  - `/` : division
- **Logiques** :
  - `ET` : ET logique
  - `OU` : OU logique
  - `NON` : Négation
- **Comparaison** :
  - `=` : égalité
  - `!=` : inégalité
  - `<, <=` : inférieur, inférieur ou égal
  - `>, >=` : supérieur, supérieur ou égal

4.2 Priorité des Opérateurs

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

Catégorie	**Opérateur(s)**
Logique	OU
Logique	ET
Logique	NON
Comparaison	<, <=, >, >=, =, !=

Arithmétique	`+, -`
Arithmétique	`*, /`

5. Entrées et Sorties

- **Afficher un message ou une variable** :
  - Syntaxe **:** `affiche(<expression>)`
  - Exemple : `affiche("Salut tout le monde !")`

- **Lire une valeur de l'utilisateur** :
  - Syntaxe **:** `lire(<variable>)`
  - Exemple : `lire(age)`

6. Exemple Complet d'un Programme en EasyCode

```
DEBUT

        --Déclaration des variables ##
        NUM : age
        REAL : salaire
        FIXE NUM : MAX_AGE = 65

        EXECUTION {
                affiche("Entrez votre âge :")
                lire(age)

                SI (age < MAX_AGE) ALORS {
                        affiche("Vous pouvez encore travailler.")
                } SINON {
                        affiche("Vous êtes en âge de retraite.")
```

```
                }

                compteur <- 0
                TANTQUE (compteur < 5) FAIRE {
                        affiche("Compteur TANTQUE : ", compteur)
                compteur <- compteur + 1
                }
        }

FIN
```

7. Analyse Lexicale avec l'outil FLEX :

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

8. 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.

- ***Les routines sémantique pour les erreurs suivantes :***

    - Variable non déclaré
    - Double déclaration
    - Incompatibilité de type
    - Division sur 0 dans le cas d'une constante.
    - Modification de la valeur d'une constante.
    - Dépassement de la taille d'un tableau.

9. Gestion de la Table des 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 implémentée sous forme d'une table de hachage. Elle 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.

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:

Erreur <type> à la ligne <numéro_ligne>, colonne <numéro_colonne>

Exemple :

Erreur Syntaxique à la ligne 5, colonne 10