## 1. Introduction

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

## 2. Description du Langage

### La structure générale

```
#PartieDeclarationBibliotheque
Programme  nomProgramme
{
      PartieDeclarationDeVariable
      PartieInstructions
}
```

### La partie de déclaration de bibliothèques

- Le programme peut contenir une liste de bibliothèque qu'on doit déclarer au début de programme.
- Chaque bibliothèque doit être sur une ligne séparée et précédée par le symbole ##.
- La liste des bibliothèques autorisées est :

| Nom | Usage | Exemple |
|---|---|---|
| **PROCESS** | Pour les instructions arithmétiques | ## PROCESS |
| **LOOP** | Pour les instructions de boucle | ## LOOP |
| **ARRAY** | Lorsqu'on utilise des tableaux | ## ARRAY |

Chaque instruction doit se terminer par le symbole $

### La Partie déclaration de variables

La partie déclaration des variables doit commencer par le mot **VAR**

La déclaration d'une variable a la forme suivante:

```
TYPE :: liste_des_variables $
```

Nous pouvons déclarer dans ce langage des variables (simples ou tableaux) et des constantes.

- **Variable simple : TYPE ::** Liste de Variables $
- **Tableau : TYPE ::** NomVariable [taille] $
- **Constante : CONST TYPE ::** Liste de Constantes $

**Liste de Variables** est une suite d'identificateurs séparés par des //
Exemple : **INTEGER Ab10 // Ae // T[5]**

**Taille d'un tableau** est un chiffre strictement positif.
**TYPE** est le nom du type de la variable qui peut être : **INTEGER, REAL. CHAR, STRING**
**Un identificateur** est une suite alphanumérique qui commence par lettre **Majuscule** et qui ne dépasse pas 10 caractères.

La liste des constantes déclarées après le mot clé **CONST** doivent être initialisés

Exemple: **CONST INTEGER** :: Xe=15 // Acad= 2020$

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

| **Le type** | **Description** | **exemple** |
|---|---|---|
| INTEGER | Un entier est une suite de chiffres. Signé ou non. Sa valeur est entre -32768 et 32767. | 5, (-5) |
| REAL | 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. | 5.5, (-5.5) |
| CHAR | Une variable de type CHAR représente un caractère.<br>Le caractère doit être mis entre deux apostrophes.<br>Exemple: CHAR Sys= 's'$ | 'A' |
| STRING | Une variable de type STRING représente une chaîne de caractères.<br>Une chaine de caractère doit etre mise entre guillemets<br>**Example:** String Section="L3-ACAD"<br>Une chaine de caractère peut contenir des lettres, chiffre ou des symboles | "abc" |

## La Partie instruction

Dans notre langage, **SEULES** les instructions suivantes sont autorisées :

- **Affectation**
- **Entrée/ sortie**
- **Boucle**
- **Conditions**

| **Instruction** | **1Description** | **Exemple** |
|---|---|---|
| Affectation | Idf := expression arithmétique | A :=2 $<br>A :=C+D$<br>Tab[2] :=A+B/2$<br>A := (-6)$ |
| Entrée/ Sortie | **Entrée :** instruction de lecture. On ne peut lire qu'une seule variable à la fois.<br>**Syntaxe :**<br>**READ** ("*signe de formatiage* " \| @ idf )<br>**Sortie :** Instruction d'affichage. On peut afficher une ou plusieurs variables<br>**Syntaxe 1 :**<br>**WRITE** (" voilà la valeur de idf *signe de formatage* " \| idf)<br>**Syntaxe 2 :**<br>**WRITE** ("idf1 idf2 *signe de formatage signe de formatage* " \| idf1 // idf2)<br>*signe de formatage* ***peut être :***<br>• **;** : si l'idf est un **INTEGER**<br>• **%** : si l'idf est un **REAL**<br>• **?** : si l'identificateur est un **STRING**<br>• **&** si l'identificateur est un **CHAR** | |
| Boucle | **While (** cond)<br>**{**<br>Bloc Instructions<br>**}**<br>**Cond :** c'est une instruction mise entre parenthèses et qui utilise des opérateurs logiques<br>**Exemple :**<br>I INF J<br>I SUP (-3)<br>I+5 SUP J | While (I INF J)<br>{<br>A :=2 $<br>A :=C+D /(-8)$<br>}<br>$ |

| | | |
|---|---|---|
| Condition | **EXECUT** INSTRUCTIONS<br>**IF (**condition )<br>**END_IF** | EXECUT I :=5 $ IF (J EG 2)<br>**END_IF**<br>$<br><br>EXECUT<br>X=A+(-3) $<br>I :=5 $<br>IF (J EG 2)<br>**END_IF**<br>**$** |
| | **EXECUT** INSTRUCTIONS<br>**IF (**condition )<br>**ELSE**<br>**EXECUT** INSTRUCTIONS<br>**END_IF** | EXECUT<br>X=A+(-3) $<br>I :=5 $<br>IF (J EG 2)<br>ELSE<br>EXECUT<br>X=A6*(-3) $<br>**END_IF**<br>**$** |

- **Les expressions arithmétiques** sont composées des opérateurs : +, *,- ,/
- **Opérateurs de comparaison sont : EG, INF , SUP, INFE, SUPE, DIFF**

## Associativité et Priorité des opérateurs

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

| **Opérateur** | | **Associativité** |
|---|---|---|
| *Opérateurs de comparaison* | SUP (>)<br>SUPE (≥)<br>EG (==)<br>DIFF (!=)<br>INFE (<=)<br>INF (< ) | Gauche |
| *Opérateurs Arithmétiques* | + - | Gauche |
| | * / | Gauche |

## Les commentaires

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

**Exemple** : /*ceci est

Un commentaire*/

## 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 implémenter l'analyseur syntaxique-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.

- **Analyse sémantique :** Parmi les erreurs à identifier pouvant citer :
  - Idf non déclaré
  - Idf double déclarée
  - Non compatibilité de type
  - Dépassement de la taille d'un tableau
  - Absence d'une bibliothèque nécessaire

- **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.

  La table doit avoir au minimum les champs suivants :
  - Nom : l'identificateur qui indique le nom de la variable (ou constante), Tab, etc.
  - Type : le type de la variable ou la constante

- Taille : la taille du tableau (la taille est égale à 1 pour les variables simples).

- **Génération de code intermédiaire** sous forme de quadruplet. Et ceci pour toutes les instructions possible, en prenant en compte les boucles imbriquées.

- **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.

**Bon courage.**