Réalisation d'un compilateur pour un mini langage Avec les Outils FLEX et BISON

I. Introduction

Le but de ce projet est de réaliser un mini-compilateur d'un langage

Description du Langage

1. La structure générale

Program NOM_PROGRAMME
PDEC
PartieDeclaration
PINST
Begin
PartieInstructions
End

Cette partie est réservée à la déclaration de toutes les variables et constantes nécessaires pour le programme.

2.1 Déclaration de variables

La syntaxe de déclaration d'une variable est la suivante :

NomVariable : Type;

NomVariable est un **identificateur** (voir section 2.3)
Type est le nom du type de la variable (voir section 2.4)
La déclaration d'un ensemble de variables d'effectue en séparant les noms de variables par ' **|**'.

NomVariable1| NomVariable2 : Type;

Exemple : **x|y|z :Pint ;**

2.2 Déclaration d'une constante :

La syntaxe de déclaration d'une constante est la suivante :

@define Type nomConstante =Valeur ; **Exemple :** @define Pint x=5 ;

2.3 Identificateur

Un identificateur est une suite alphanumérique (lettre minuscules et chiffres) et/ou les tirets « _», qui commence par lettre et qui ne dépasse pas 12 caractères et il ne doit pas de terminer par un tiret « _ ». Le nom de programme et les noms de variables des constantes sont des identificateurs.
Exemple : a, a1, s12, mon_program.

2.4 Les types

Le langage accepte les types suivants :

Le type	Description	exemple
Pint	Un entier est une suite de chiffres. Signé ou non. Sa valeur est entre -32768 et 32767.	5, -5
Pfloat	Une suite de chiffres contenant le point décimal.	5.5, -5.5

3. La Partie instruction

Toute instruction doit se terminer par une point-virgule. Les instructions sont :

Instruction	Description	Exemple
Affectation	Idf ← valeur; Idf ← expression arithmétique ;	A ←2 ; A --< C+D ;
Boucle	**FOR** i<-- val **WHILE** val **DO** Bloc Instructions **ENDFOR**	**FOR** i←1 **WHILE** 5 **DO** J←j/2 ; C←c+j ; **ENDFOR**
condition	**DO** Bloc Instructions1 **IF: (**condition) **ELSE** Bloc Instructions2	do i←5 : if (j==2)

Remarque : on peut avoir les instructions imbriquées : **FOR** i<-- 1 **WHILE** 5
DO
DO i←5: **IF** (j==2)
ELSE J←6
ENDFOR

3.1 Les expressions

Les expressions peuvent être arithmétiques, de comparaison ou logique. Pour les composer, nous utilisons les opérations indiqués dans le tableau suivant :

Type d'expression	Les opérations	Exemple d'expression
Expression Arithmétique	Addition : + Soustraction : - Multiplication : * Division : /	1*2 , 1.2-6.1,1/3*(a+2) ; etc
Expression de comparaison	Supérieur :> Inferieur :< Egal : == Diffèrent : != Supérieur ou égal : >= Inférieur ou égal :<=	x>1, y==x+1, (x<=1.2*3)
Expression logique	Le et logique :& Le ou logique |	(X<3 &y==1),((x>1) |(x<5))

Associativité et Priorité des opérateurs

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

Opérateur logiques	|(ou)	Priorité ↓
	&(et)	
	!(négation)	
Operation de comparaison	> ,>=,== , != ,<= ;<	
Opérateurs Arithmétiques	+ -	Priorité ↓
	* /	

II.9 Les commentaires

Un commentaire peut être écrit sur une seule ligne en précédant le texte par « // » ou sur plusieurs lignes en mettant le texte entre « /* » et « */ »

Exemple : // ceci est un commentaire

/* ceci est

Un commentaire*/

II. Travail à réaliser :

Ci-dessous les différentes briques à implémenter afin de réaliser le compilateur demandé

1. Analyse lexicale avec l'outil FLEX:

Le but de l'analyseur lexical est de reconnaitre chaque mot d'un code écrit en un langage donné et de lui associer la catégorie lexicale à laquelle il appartient. Afin de réaliser cet analyseur lexical, nous utilisons l'outil FLEX. Un manuel est mis à disposition pour apprendre l'essentiel sur cet outil.

2. Traduction dirigée par la syntaxe avec l'outil Bison

L'analyse lexicale est suivie par l'analyse syntaxico-sémantique et la génération du code intermédiaire.

- L'analyse syntaxique consiste à vérifier la syntaxe du code source selon la définition de sa grammaire.
- L'analyse sémantique intervient ensuite afin d'effectuer les vérifications nécessaires à la sémantique du langage et met à jour la table des symboles.
- Enfin un code intermédiaire est généré sous forme de quadruplets.

Pour réaliser cette étape, nous utilisons l'outil BISON. Cet outil prend en entrée une spécification de la grammaire du langage, les règles de priorités ainsi que les routines sémantiques. L'outil BISON produit un analyseur ascendant qui opère sur des grammaires LALR. Les routines sémantiques doivent assurer les vérifications concernant la résolution des noms (exp : association IDF-TYPE), La vérification des types, etc.

Le code intermédiaire généré doit être sous forme de quadruplets. Par exemple, l'instruction « x=3 » est traduite par le quadruplet « (=, 3,, x) ».

3. Gestion de la table de symboles (TS):

La table des symboles regroupe l'ensemble des variables et des constantes définies dans le code source 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. La table doit être implémentée sous forme de liste chainée. Il est demandé de prévoir des fonctions (écrite en langage C) permettant la recherche et

l'insertion des éléments dans la table des symboles. Chaque élément de la TS doit avoir les champs suivants:

-**Nom** : l'identificateur qui indique le nom de la variable (ou constante).
-**Nature** : variable ou constante.
-**Type** : le type de la variable ou la constante.

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, syntaxique ou sémantique est détectée par le compilateur, elle doit être signalée le plus précisément possible, par sa nature et sa localisation dans le fichier source selon le format suivant:

Type_ de_ l'erreur, Num_ ligne, Num_colonne : entité qui a généré l'erreur.