Projet Compilation

Introduction

Ce projet consiste à implanter les composants de base d'un compilateur pour un langage proche d'un sous-ensemble de Pascal. Ce mini-Pascal doit permettre d'exprimer des programmes élémentaires tels que le programme suivant :

```
program exemple ; (* programme factoriel *)
var x,y : LongInt ;
   func fact (n : LongInt) : LongInt ;
      {
      if a=0 then {i}
      {fact := n*fact(n-1) }
      } ;
   {
   read (x) ;
   y :=x**x ;
   write(fact(x)) ;
   write(y)
   }.
```

Ce langage doit permettre d'utiliser :

- Les types simples
- Les expressions booléennes et arithmétiques
- La conditionnelle
- L'itération limitée exclusivement à l'instruction repeat … until …
- La déclaration et l'appel de fonction. La récursivité est permise et les paramètres sont passés par valeur.

Syntaxe du langage

Nous décrivons dans cette section la grammaire de notre mini-Pascal.

programme	→	**program id ;** déclarations déclaration_sous_programmes instruction_composée **.**
liste_identificateurs	→ |	**id** liste_identificateurs **, id**
déclarations	→ |	**var** déclaration ε

déclaration	→	liste_identificateurs **:** type **;** déclaration
	|	liste_identificateurs **:** type **;**
type	→	**Int**
	|	**LongInt**
déclaration_sous_programmes	→	déclaration_sous_programmes déclaration_sous_programme **;**
	|	ε
déclaration_sous_programme	→	entete_sous_programmes
	|	déclarations
	|	instruction_composée
entete_sous_programmes	→	**func id** arguments **:** type **;**
	|	**proc id** arguments **;**
arguments	→	**(** liste_paramètres **)**
	|	E
liste_paramètres	→	liste_identificateurs **:** type
	|	liste_paramètres **;** liste_identificateurs **:** type
instruction_composée	→	**{** instruction_optionnelles **}**
instruction_optionnelles	→	liste_instructions
	|	ε
liste_instructions	→	instruction
	|	liste_instructions **;** instruction
instruction	→	variable **opaffect** expression
	|	instruction_proc
	|	instruction_composée
	|	**if** expression **then {** instruction **}**
	|	**if** expression **then {** instruction **} {** instruction **}**
	|	**repeat** instruction **until** expression
variable	→	**Id**
instruction_proc	→	**id**
	|	**id (** liste_expressions **)**
liste_expressions	→	expression
	|	liste_expressions **,** expression
expression	→	expression_simple
	|	expression_simple **oprel** expression_simple
expression_simple	→	terme
	|	signe terme
	|	expression_simple **opadd** terme
terme	→	facteur
	|	terme **opmul** facteur
	|	facteur ****** terme
facteur	→	**id**
	|	**id (** liste_expressions **)**
	|	**nb**
	|	**(** expression **)**
	|	**not** facteur
signe	→	+
	|	-

Conventions lexicales

- Les commentaires sont entourés par (* **et** *) . Ils ne peuvent pas contenir (* . Un commentaire peut apparaître après une unité lexicale quelconque.
- Les blancs entre les unités lexicales sont optionnels, excepté pour les mots clés qui doivent entourés par des blancs, des fins de lignes ou le point final.
- L'unité lexicale **id**, représentant les identificateurs, est formée d'une lettre suivie de lettres ou de chiffres :

  lettre → [a –z A-Z]
  chiffre → [0 – 9]
  id → lettre (lettre | chiffre)*

- L'unité lexicale nb correspond aux entiers non signés :

  nb → chiffre chiffre*

- Les mots clés sont réservés et ils apparaissent en gras dans la grammaire.
- Les opérateurs relationnels (oprel) sont = , <> , < , <= , > , >= .
- Les opérateurs opadd sont +, - , or .
- Les opérateurs opmul sont * , / , div , mod , and
- Le lexème de l'unité lexicale opaffect est **:=**
- Le lexème de l'unité lexicale ** est l'opérateur de puissance.

Travail demandé

Ce projet consiste à écrire un compilateur pour le langage défini ci-dessus. Il consiste à faire l'analyse lexicale et syntaxique à l'aide des outils Flex et Bison

1. Ecrire l'analyseur lexical
2. Ecrire l'analyseur syntaxique