Le 21 avril 2019

Ecole Nationale des Sciences de l'Informatique

Ferchichi Mohamed Kacem & Gallala Souha & Gaaloul Fares

[DOCUMENTATION DU PROJET DE COMPILATION]

I. Partie 1:

1. Définition des unités lexicales :

UNITE LEXICALE	CODE	UNITE LEXICALE	CODE
PROGRAM	0	Nb	16
VAR	1	OPEREL	17
INTEGER	2	OPADD	18
CHAR	3	OPMUL	19
BEGIN	4	OPAFF	20
END	5	SEMCOL	21
IF	6	PT	22
THEN	7	DBPT	23
ELSE	8	COMMA	24
WHILE	9	OPP	25
DO	10	CLP	26
READ	11	DOLLAR	27
READLN	12		
WRITE	13		
WRITELN	14		
ID	15		

2. Construction des tables et ecriture d'analyseur lexical :

- On va coder un analyseur lexical qui lit tous les mots en eliminant les caracteres separateurs et les ranger dans une table des symboles en se basant en se basant sur la table des identificateurs.
- La table des identificateurs est une structure dont les attributs sont :

- id : entier
- name : chaine de caracteres
- type : entier
- val : entier

- La table des symboles est une structure de donne`es avec les attributs :

- type : entier
- attr : entier
- line : entier

⇨ D'où l'ecriture de l'analyseur lexical (voir code)

II. Partie 2 :

1. Nettoyage de la grammaire :

1.1 Elimination de la récursivité à gauche :

- Les règles de production contenant une récursivité à gauche sont :

  - **Dcl → Dcl var Liste_id : Type ; | ε**
    Devient :
    - Dcl → Dcl'
    - Dcl'→ var Liste_id : Type ; Dcl' | ε

    Dcl → var Liste_id : Type ; Dcl | ε

  - **Liste_id →id | Liste_id , id**
    Devient :
    - Liste_id → id Liste_id'
    - Liste_id' → , id Liste_id' | ε

  - **Liste_inst → I | Liste_inst ; I**
    Devient :
    - Liste_inst → I Liste_inst'
    - Liste_inst' → ; I Liste_inst' | ε

  - **Exp_simple → Terme | Exp_simple opadd Terme**
    Devient :
    - Exp_simple → Terme Exp_simple'
    - Exp_simple' → opadd Terme Exp_simple' | ε

  - **Terme → Facteur | Terme opmul Facteur**
    Devient :
    - Terme → Facteur Terme'
    - Terme' → opmul Facteur Terme' | ε

2.2 Factorisation à gauche :

  - **Exp → Exp_simple | Exp_simple oprel Exp_simple**
    Devient :
    - Exp → Exp_simple Exp'
    - Exp' → ε | oprel Exp_simple

- La grammaire alors devient :

  - **P → program id ; Dcl Inst_composé .**
  - **Dcl → var Liste_id : Type ; Dcl | ε**
  - **Type → integer | char**
  - **Inst_composée → begin Inst end**
  - **Inst → Liste_inst | ε**
  - **Liste_inst → I Liste_inst'**
  - **Liste_inst' → ; I Liste_inst' | ε**
  - **Liste_id → id liste_id'**
  - **Liste_id' → , id Liste_id' | ε**
  - **I → Id := Exp_simple | if Exp then I else I | while Exp do I | read (id) | readln (id) | write (id) | writeln (id)**
  - **Exp → Exp_simple Exp'**
  - **Exp' → oprel Exp_simple | ε**
  - **Exp_simple → Terme Exp_simple'**
  - **Exp_simple' → opadd Terme Exp_simple' | ε**
  - **Terme → Facteur Terme'**
  - **Terme' → opmul Facteur Terme' | ε**
  - **Facteur → id | nb | (Exp_simple)**

2. Preparation de la table d'analyse M :

2.1. Les premiers :

- **Premier (Facteur)** = premier (nb) U premier (id) U premier (Exp-simple)
  = { nb, id ,(}
- **Premier (Terme')** = premier (opmul facteur terme') U {ε}
  = {opmul, ε}
- **Premier (Terme)** = premier (Facteur Terme') = premier (Facteur)
  ={ nb, id, (}
- **Premier (Exp_simple')** = premier (opadd Terme Exp_simple') U {ε}
  = { opadd , ε }
- **Premier (Exp_simple)** = premier (Terme) = { nb,id,(}
- **Premier (Exp')** = premier(operel Exp_simple) U {ε} = {operel, ε}
- **Premier (Exp)** = premier (Exp_simple Exp')= premier (Exp_simple)
  = {nb, id, (}
- **Premier (I)** = premier (id := Exp_simple) U premier (if Exp then I else I) U premier (while Exp do I) U premier (read (id)) U premier (readln (id)) U

premier (write (id)) U premier (writeln (id)) = { id, if, while, read, readln, write, writeln }

- **Premier (Liste_id')** = premier (, id liste_id') U { ε} = {,} U {ε}
- **Premier (Liste_id)** = premier(id liste_id') = {id}
- **Premier (Liste_Inst')** = premier(; I liste_Inst') U {ε} = { ;} U {ε}
- **Premier (Liste_Inst)** = premier (I Liste_Inst') = premier (I) = { id, if, while, read, readln, write, writeln}
- **Premier (Inst)** = premier (liste_Inst) U {ε} = { id, if, while, read, readln, write, writeln, ε}
- **Premier (Inst_composée)** = premier (begin Inst end) = { begin }
- **Premier (Type)** = { integer, char }
- **Premier (Dcl)** = premier (var liste_id : Type ; Dcl) U {ε} = { var, ε }
- **Premier (P)** = premier (program id ; Dcl Inst_composée) = { program }

2.2. Les suivants :

- **Suivant (P)** = { $ }
- **Suivant (Dcl)** = premier (Inst_composée) = { begin }
- **Suivant (Type)** = { ; }
- **Suivant (Inst_composé)** = { . }
- **Suivant (Inst)** = { end }
- **Suivant (Liste_Inst)** = Suivant (Inst) = { end }
- **Suivant (Liste_Inst')** = Suivant (Liste_Inst) = { end }
- **Suivant (Liste_id)** = { : }
- **Suivant (Liste_id')** = Suivant (Liste_id) = { : }
- **Suivant (I)** = premier (Liste_Inst')\ { ε} U {else} U suivant (List_Inst')
  = { ;, else, end}
- **Suivant (Exp)** = { then, do }
- **Suivant (Exp')** = suivant (Exp) = {then, do }
- **Suivant (Exp_simple)** = suivant (I) U {) } U premier (Exp')\ {ε} U suivant (Exp') = { ; , else,), operel, then, do, end}
- **Suivant (Exp_simple')** = suivant (Exp_simple) = { ; , else,), operel, then, do, end}
- **Suivant (Terme)** = premier (Exp_simple')\{ ε} U suivant (Exp_simple)
  = { opadd, ;,), else, operel, then, do, end }
- **Suivant (Terme')** = Suivant (Terme) = { opadd, ;,), else, operel, then, do, end }
- **Suivant (Facteur)** = premier(Terme') \ { ε} U suivant (Terme) = { opmul, opadd, ;,), else, operel, then, do, end }

LES NON TERMINAUX	LES PREMIERS	LES SUIVANTS
P	{ program }	{ $ }
DCL	{ var, ε }	{ begin }
TYPE	{ integer , char }	{ ; }
INST_COMPOSEE	{ begin }	{ . }
INST	{id,if,while,read,readln,write,writeln} U { ε }	{ end }
LISTE_INST	{id,if,while,read,readln,write,writeln}	{ end }
LISTE_INST'	{ ε , ; }	{ end }
LISTE_ID	{ id }	{ : }
LISTE_ID'	{ ε } U { , }	{ : }
I	{id,if,while,read,readln,write,writeln}	{ ; , else, end }
EXP	{ nb, id, (}	{ then, do }
EXP'	{ ε, operel }	{ then, do }
EXP_SIMPLE	{ nb, id , (}	{ ;, else,), operel, then , do, end}
EXP_SIMPLE'	{ opadd, ε }	{ ;, else,), operel, then, do, end}
TERME	{ nb, id, (}	{ opadd }
TERME'	{ opmul, ε }	{opadd, ;,), else, operel, then, do, end}
FACTEUR	{ nb, id, (}	{opmul,opadd, operel, then, do, else, end, ;}

⇨ On peut maintenant effectuer une analyse syntaxique par descente prédictive (voir code)

Realise` par :
Ferchichi Mohamed Kacem
Gallala Souha
Gaaloul Fares
Groupe II1C.