4. COMPILATION ET INTERPRETATION DES INSTRUCTIONS CONDITIONNELLE ET REPETITIVE.

L'objectif est de compiler et interpréter des programmes sources écrits dans le langage mini-Pascal dont la syntaxe du langage est complétée par:
- les instructions conditionnelles du type si-alors et si-alors-sinon;
- les instructions répétitives du type tantque-faire.

⇨ **L'instruction conditionnelle**
Les instructions conditionnelles du langage mini-Pascal sont de la forme:
SI exp_arithmétique ALORS instruction_ou_bloc
ou: SI exp_arithmétique ALORS instruction_ou_bloc SINON instruction_ou_bloc
Si la valeur de l'expression arithmétique est différente de zéro, alors c'est le bloc d'instructions (ou l'instruction) suivant le mot-clé réservé ALORS qui est exécuté. Le bloc d'instructions (ou l'instruction) suivant le mot-clé SINON, s'il y en a un est ignoré.
Au contraire, si la valeur de l'expression arithmétique est égale à zéro, c'est le bloc d'instructions (ou l'instruction) suivant le mot-clé SINON qui est exécuté, s'il existe.

Exemple1:
```
SI resultat ≠0 ALORS
DEBUT
        ECRIRE( 'Le résultat est différent de zéro');
        ECRIRE();
        ECRIRE( 'Il est égal à :', resultat);
FIN
SINON
        ECRIRE( 'Le résultat est nul');
```

⇨ **L'instruction répétitive**
Les instructions répétitives du langage mini-Pascal sont de la forme:
TANTQUE exp_arithmétique FAIRE instruction_ou_bloc
Tant que la valeur de l'expression arithmétique est différente de zéro, l'instruction ou le bloc d'instructions suivant le mot-clé réservé FAIRE est exécuté.

Exemple2:
```
{ affichage des 10 chiffres dans l'ordre décroissant }
i := 9;
TANTQUE i ALORS
DEBUT
        ECRIRE( i, ' ');
        i := i-1;
FIN
ECRIRE( '0');
ECRIRE();
```

⇨ **Nouveaux mots-clés réservés**
Il faut rajouter les mots-clés SI, ALORS, SINON, TANTQUE et FAIRE au tableau des mots-clés et penser à augmenter le nombre de mots-clés réservés de 5, soit:
CONST nb_mots_reserves = 12;

⇨ **Modification de la grammaire du langage mini-Pascal**
La grammaire G_1 du langage mini-Pascal est définie à partir de la grammaire précédente G_0. Elle est maintenant définie par le quadruplet (T_1, N_1, PROG, P_1) où:

- T_1 est l'ensemble des symboles terminaux:
  $T_1 = T_0 \cup$ { 'SI', 'ALORS', 'SINON', 'TANTQUE', 'FAIRE'}

- N_0 est l'ensemble des symboles non terminaux:
  $N_1 = N_0 \cup$ { INST_NON_COND, INST_COND, INST_REPE }

- PROG est le symbole non terminal initial (PROG $\in N_1$)
- P_1 est l'ensemble des règles de production, c'est l'ensemble des règles de production de P0 dans lequel on remplace la règle: INSTRUCTION → AFFECTATION | LECTURE | ECRITURE | BLOC

par les règles:
```
INSTRUCTION     → INST_NON_COND | INST_COND
INST_NON_COND   → AFFECTATION | LECTURE | ECRITURE | BLOC | INST_REPE
INST_REPE       → 'TANTQUE' EXP 'FAIRE' INSTRUCTION
INST_COND       → 'SI' EXP 'ALORS' (  INST_COND
                                    | INST_NON_COND { 'SINON' INSTRUCTION } )
```

1. Analyse syntaxique et sémantique

■ **Ecriture des fonctions associées à chaque nouveau non terminal de la grammaire**

1°/ Analyse syntaxique: Il faut réécrire la fonction booléenne récursive INSTRUCTION et écrire les fonctions booléennes récursives INST_NON_COND, INST_REPE et INST_COND.
2°/ Analyse sémantique: Il n'y a pas de contraintes sémantiques supplémentaires à vérifier liées aux instructions conditionnelles ou répétitives.

2. Génération de code et interprétation

Les instructions de la machine virtuelle doivent être complétées avec les instructions de contrôle de flot suivantes: ALLE (aller à), ALSN (aller si nul). Ce sont des instructions de rupture de séquence qui nécessitent une adresse comme opérande. Seule l'instruction ALSN manipule le sommet de la pile.

Instructions du P_CODE	Interprétation	
ALLE op	Instruction de branchement inconditionnel: Aucune condition quant à l'état de la pile d'exécution n'est nécessaire. L'instruction suivante qui sera exécutée est celle qui se trouve à l'adresse donnée par op.	CO := op;

ALSN op	Instruction de branchement conditionnel:
	La pile d'exécution doit avoir à son sommet une valeur à tester. Si cette valeur est nulle, alors l'instruction suivante qui sera exécutée est celle qui se trouve à l'adresse donnée par op. Sinon, la valeur au sommet de la pile est non nulle et l'instruction suivante sera exécutée. La valeur au sommet de la pile est dépilée.
		if PILEX[SOM_PILEX] = 0 then
		  CO := op
		else
		  CO := CO + 1;
		SOM_PILEX := SOM_PILEX - 1;

Exemples: Pour la partie de programme du premier exemple de la page précédente, le code machine suivant doit être généré:

```
    EMPI @resultat
    CONT
    EMPI 0
    SOUS
    ALSN x
    ECRC 'L' 'e' ' ' 'r' 'é' 's' 'u' 'l' 't' 'a' 't' ' ' 'e' 's' 't' ' ' 'd' 'i' 'f' 'f' 'é' 'r' 'e' 'n' 't' ' ' 'd' ' 'e' ' ' 'z' 'é' 'r' 'o' FINC
    ECRL
    ECRC 'l' 'l' ' ' 'e' 's' 't' ' ' 'é' 'g' 'a' 'l' ' ' 'à' ' ' ':' FINC
    EMPI @resultat
    CONT
    ECRE
    ALLE y
x:  ECRC 'L' 'e' ' ' 'r' 'é' 's' 'u' 'l' 't' 'a' 't' ' ' 'e' 's' 't' ' ' 'n' 'u' 'l' FINC
y:  ...
```

Pour la partie de programme du second exemple de la page précédente, le code machine suivant doit être généré:

```
        EMPI @I
        EMPI 9
        AFFE
deb:    EMPI @i
        CONT
        ALSN fin
        EMPI @i
        CONT
        ECRE
        ECRC ' ' FINC
        EMPI @i
        EMPI @i
        CONT
        EMPI 1
        MOIN
        AFFE
        ALLE deb
fin:    ...
```

■ **Fonction associée au non terminal INST_COND de la grammaire**

Exemples: si i alors i:=1 sinon i:=2;

```
        EMPI @i
        CONT
        ALSN x
        EMPI 1
        EMPI 1
        AFFE
        ALLE y
x:      EMPI 1
        EMPI 2
        AFFE
y:      ...
```

si 1 alors ECRIRE();

```
        EMPI 1
        ALSN x
        ECRL
        ALLE y
x,y:    ...
```

- Le code de l'expression arithmétique est généré lors de l'appel de la fonction EXP
- Après la reconnaissance de l'expression arithmétique, il faut produire l'instruction ALSN. A cet instant, on ne connaît pas l'adresse dans le PCODE de la fin du bloc d'instructions (ou de l'instruction) « alors ». Il faut mémoriser la valeur de CO correspondant à cette adresse dans une pile pour pouvoir la remplir ultérieurement. On peut conserver ces valeurs de CO dans la même pile que celle utilisée pour mémoriser les opérateurs, PILOP:

```
PCODE[CO]:=ALSN;
SOM_PILOP:= SOM_PILOP+1;
PILOP[SOM_PILOP]:=CO+1;
CO:=CO+2;
```

- Si l'instruction conditionelle I est:
  - de la forme « si EXP alors INST_COND », nous noterons I1 l'instruction conditionelle qui forme I,
  - de la forme « si EXP alors INST_NON_COND », nous noterons I2 l'instruction non conditionelle qui forme I,
  - de la forme « si EXP alors INST_NON_COND sinon INSTRUCTION », nous noterons I3 et I4 l'instruction non conditionelle et l'instruction qui forment I,

1er cas: le code de l'instruction conditionelle I1 est généré lors de l'appel de la fonction INST_COND.
2e cas: le code de l'instruction non conditionelle I2 est généré lors de l'appel de la fonction INST_NON_COND.
3e cas: le code de l'instruction non conditionelle I3 est généré lors de l'appel de la fonction INST_NON_COND.
Après la reconnaissance de l'instruction (I1, I2 ou I3), il faut produire l'instruction ALLE en mémorisant l'adresse du CO suivant pour l'opérande de ALLE (dont on ne connaît pas la valeur à cet instant) et affecter l'opérande de l'instruction ALSN produite précédemment. La position de l'opérande de ALSN est mémorisée dans PILOP, il faut la dépiler:

```
PCODE [PILOP[SOM_PILOP]]:=CO+2;
SOM_PILOP:= SOM_PILOP-1;
PCODE[CO]:=ALLE;
SOM_PILOP:= SOM_PILOP+1;
PILOP[SOM_PILOP]:=CO+1;
CO:=CO+2;
```

- A la fin de la reconnaissance de l'instruction conditionelle I, il faut affecter l'opérande de l'instruction ALLE produite précédemment. La position de cet opérande est mémorisée dans PILOP, il faut la dépiler:

```
PCODE [PILOP[SOM_PILOP]]:=CO;
SOM_PILOP:= SOM_PILOP-1;
```

■ **Fonction associée au non terminal INST_REPE de la grammaire**

- Il faut mémoriser le CO du début de l'instruction tant que:

```
PCODE [PILOP[SOM_PILOP]]:=CO;
SOM_PILOP:= SOM_PILOP-1;
```

- Le code de l'expression arithmétique est généré lors de l'appel de la fonction EXP.

- Apres la reconnaissance de l'expression arithmetique, il faut produire l'instruction ALSN. A cet instant, on ne connaît pas l'adresse dans le PCODE de la fin du bloc d'instructions (ou de l'instruction). Il faut mémoriser la valeur de CO correspondant à cette adresse dans la pile PILOP:

```
PCODE[CO]:=ALSN;
SOM_PILOP:= SOM_PILOP+1;
PILOP[SOM_PILOP]:=CO+1;
CO:=CO+2;
```

- Le code de l'instruction est généré lors de l'appel de la fonction INSTRUCTION.
- Après la reconnaissance de l'instruction, il faut affecter l'opérande de l'instruction ALSN produite précédemment et produire l'instruction ALLE avec le bon opérande. La valeur de l'opérande de ALLE et la position de l'opérande de ALSN sont mémorisées dans PILOP, il faut les dépiler:

```
PCODE [PILOP[SOM_PILOP]]:=CO+2;
SOM_PILOP:= SOM_PILOP-1;
PCODE[CO]:=ALLE;
PCODE[CO+1]:= PILOP[SOM_PILOP];
SOM_PILOP:= SOM_PILOP-1;
CO:=CO+2;
```

■ **Procédures ERREUR, CREER_FICHIER_CODE et INTERPRETER**

Ces procédures doivent évidemment être modifiées de façon à prendre en compte les instructions conditionelles et répétitives.