NOM : .....
PRENOM : .....

# Examen : Traduction des langages

**1h45 avec documents**

**Remarques :** Le barème est donné à titre indicatif. Les exercices sont indépendants.

## Exercice 1 : Compréhension du cours (3pt)

Indiquer à quel(s) moment(s) (analyse lexicale ($lex$), analyse syntaxique ($syn$), analyse sémantique ($sem$), linéarisation ($lin$), sélection d'instructions ($selec$), allocation de registre ($alloc$), optimisation ($opt$), émission de code ($emis$), édition de liens ($edl$), exécution ($exec$), les actions suivantes sont réalisées : 0.25 par action complètement juste

- Action 1 : Résolution des identificateurs.
- Action 2 : Contrôle de la structure du programme.
- Action 3 : Gestion (définition, utilisation, ...) des types de l'utilisateur.
- Action 4 : Traitement des commentaires.
- Action 5 : Calcul des valeurs des variables.
- Action 6 : Construction de l'arbre abstrait.
- Action 7 : Gestion des références croisées dans des fichiers sources différents.
- Action 8 : Choix de la portée des variables.
- Action 9 : Reconnaissance des mots clés du langage.
- Action 10 : Gestion de la surcharge des opérateurs.
- Action 11 : Suppression du code mort.
- Action 12 : Choix de la branche de la conditionnelle.

| | $lex$ | $syn$ | $sem$ | $lin$ | $selec$ | $alloc$ | $opt$ | $emis$ | $edl$ | $exec$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Action 1 | | | x | | | | | | | |
| Action 2 | | x | | | | | | | | |
| Action 3 | | x | x | | | | | | | |
| Action 4 | x | | | | | | | | | |
| Action 5 | | | | | | | | | | x |
| Action 6 | | x | | | | | | | | |
| Action 7 | | | | | | | | | x | |
| Action 8 | | (x) | x | | | | | | | |
| Action 9 | x | | | | | | | | | |
| Action 10 | | | x | | | | | | | |
| Action 11 | | | | | | x | | | | |
| Action 12 | | | | | | | | | | x |

## Exercice 2 : Manipulation de grammaire (5pt)

Soit $G$ la grammaire du $\pi$-calcul, définie par : $G = (\{P\}, \{|, !, 0, (, ), \nu, id, \langle, \rangle\}, R, P)$ où $R =$

1. $P \rightarrow P|P$ (processus s'exécutant en parallèle)
2. $P \rightarrow !P$ (réplication)
3. $P \rightarrow 0$ (processusnul)
4. $P \rightarrow (\nu\ id)P$ (déclaration de canal de communication)
5. $P \rightarrow id(id).P$ (lecture sur un canal)
6. $P \rightarrow id\langle id\rangle.P$ (écriture sur un canal)

1. La grammaire est-elle algébrique ? LL(1) ?
   1point. Récursive gauche, ne peut pas être LL(1).
2. Donner une dérivation pour le mot $(\nu\ c)\ c\langle x\rangle.0|c(y).y\langle z\rangle.0$
   1.5 points
3. Donner l'arbre de dérivation correspondant.
   1.5 points
4. La grammaire est-elle ambiguë ?
   1 point. La règle 1 fait qu'elle l'ait. cf la grammaire des expressions

## Exercice 3 : Écriture de grammaire (3pt)

**.properties** est une extension de fichier essentiellement utilisée en Java et qui permet aux technologies compatibles de stocker les paramètres de configuration d'un logiciel.
Chaque paramètre est stocké comme une paire composée d'un pointeur de référence ('key') donnant le nom du paramètre, suivi de la valeur désirée de ce paramètre. Le pointeur de référence et la valeur peuvent être séparés par ":" ou par "=".
Chaque ligne d'un fichier .properties comporte normalement une seule paire.
Les fichiers .properties peuvent utiliser le (#) ou le (!) comme premier caractère de ligne afin de mettre en commentaire le reste de la ligne qui devient un texte banalisé.

1. Inventer un fichier .properties simple qui contient au moins une ligne en commentaire et qui donne les valeurs de deux paramètres.
   1 point
   # commentaire
   nb : 2
   op = +
2. Donner une grammaire qui permet d'engendrer les fichiers .properties.
   2 points
   $F \rightarrow L\ F$
   $F \rightarrow L$
   $L \rightarrow \#commentaire$
   $L \rightarrow !commentaire$
   $L \rightarrow id = val$
   $L \rightarrow id : val$

## Exercice 4 : Ajout d'un opérateur (7pt)

Nous souhaitons compléter notre langage RAT, vu en cours, avec les opérateurs $++$ (resp. $--$) qui incrémente (resp. décrémente) la valeur d'un entier puis renvoie celle-ci.

L'opérateur s'utilise de la façon suivante :

```
prog{
  int y = 2;
  int x = ++y; //x vaut 3 et y 3
  int z = (++y*3); //y vaut 4 et z vaut 12
}
```

Seul un identifiant peut être pré-incrémenté / décrémenté.

1. Quelles règles faut-il ajouter à la grammaire RAT ?

   1 point

   $E \rightarrow ++ id$

   $E \rightarrow -- id$

2. La grammaire RAT ainsi complétée est-elle toujours LL(1) ?

   1 point

   Oups, elle ne l'ai pas à la base !

3. Donner l'arbre abstrait correspondant au programme précédent.

   1 point

4. Donner les actions à réaliser pour traiter cette nouvelle règle. Ces actions seront données sous forme de pseudo-code (ou code Java), en précisant le type de noeud de votre AST auxquelles elles sont associées. On doit traiter :

   (a) la table des symboles ; 1 point - vérifier que id a été déclaré globalement

   (b) le typage ; 1 point - vérifier que id est un entier

   (c) la génération de code. 2 point - charger id + incrémenter/décrémenter + enregistrer nouvelle valeur

## Exercice 5 : Réflexion sur les exceptions (2pt)

Nous souhaitons ajouter un mécanisme d'exceptions à notre langage. Proposer des pistes de réflexion pour que le compilateur puisse traiter un tel mécanisme.