Documents : interdits **Durée** : 1h/1h20 t.temps **Barème** : /20 indicatif

Préambule : Définition du langage

On considère le langage `programming with numbers and letters (pnl)` dont les seules valeurs sont les nombres et les lettres de l'alphabet. Les conditions pour les boucles et conditionnelles seront implémentées avec des entiers (considérés comme vrais si différents de 0).

Voici sa syntaxe :

```

exp : Letter          inst : 'print' '(' exp ')'
    | Int             | 'if' '(' exp ')' inst
    | 'toLetter' '(' exp ')' | 'while' '(' exp ')' inst
    | 'toInt' '(' exp ')'   | Type Id
    | '(' exp '+' exp ')'   | Id '=' exp
    | 'read' '(' ')'       | '{' (inst ';' )* '}'
    | Id
                                Type : 'int' | 'letter'
Letter : \'[a-zA-Z]\
Id : [a-zA-Z][a-zA-Z0-9_-]
prog : inst+

```

Exercice 1 : Fonctions

Question 1 (/4) Modifiez la grammaire en ajoutant la gestion des fonctions dans la syntaxe (déclarations de type `int f(int x,...){ ... }`). Pour simplifier, on considèrera que les fonctions doivent toujours retourner une valeur (pas de fonction `void`). Les fonctions doivent toutes être déclarées au niveau du programme (c'est-à-dire au niveau le plus haut de l'AST : pas de fonctions internes).

Décrivez les ajouts et modifications nécessaires dans la syntaxe donnée.

Question 2 (/4) Considérez l'extrait de code suivant :

```

int f(letter c, int n){
  int i; i=n;
  while(i){
    letter s; s=c;
    if(i+n){
      int j;j=n;
      print (j+i);
    }
    if(n+n){
      letter j; j='l';
    }
  }
  return i;
}

```

Rappel les **signatures** de fonctions contiennent leur nom, leur type de retour, le nombre le nom et le type de leurs paramètres¹. La table des symboles associe chaque identifiant de fonction à sa signature².

1. Décrivez (schématiquement) la table des symboles construite à l'issue de l'analyse de ce code par le `TableBuilder`
2. Décrivez les états successifs de la pile des blocs visités. (vous pouvez identifier vos blocs sur le sujet, pensez alors à le rendre avec votre copie).

¹Exemple, la signature de `f` peut être représentée comme: `{nom='f',retour=int,params=[('c',letter),('n',int)]}`

²Les fonctions apparaissent au niveau du programme, donc c'est une table indépendante de celle pour les variables, et n'a pas besoin de gérer les blocs.

Question 3 (/4) Écrivez les méthode de visite du `TypeChecker` pour les nœuds suivants. On suppose que le reste du programme est déjà codé comme en TP.

(1) `Id` (2) `+` (3) `if`³

Exercice 2 : Typage formel

Pour rappel, les règles de dérivation de typage prennent la forme suivante, où T est un type quelconque et Γ est un environnement de typage quelconque, c'est-à-dire une association des variables aux types. Voici quelques exemples pour `pn1` (dont les types nécessaires sont `int`, `letter`, `inst`) :

$$\frac{}{\Gamma, x : T \vdash x : T} \text{ var} \qquad \frac{\Gamma, x : T \vdash p : \text{inst}}{\Gamma \vdash \{T \ x; p\} : \text{inst}} \text{ decl}$$

$$\frac{\Gamma \vdash m : \text{int} \quad \Gamma \vdash n : \text{int}}{\Gamma \vdash m + n : \text{int}} \text{ plus} \quad \frac{\Gamma \vdash m : \text{int}}{\Gamma \vdash \text{print}(m) : \text{inst}} \text{ print}$$

Question 1 (/5) Écrivez des règles de typage cohérentes pour les constructions syntaxiques suivantes, avec les contraintes associées. (On considèrera comme en cours qu'il n'y a aucune structure de bloc dans les programmes et que l'on a toujours un seul environnement) (le typage doit bien sûr respecter la syntaxe du programme) :

1. `toInt` et `toLetter`, qui sont des opérateurs de conversion entre entiers et lettres.
2. affectation
3. `while`
4. Séquence d'instructions.

Question 2 (/3) Écrivez la dérivation de typage du programme suivant : `{int x; letter a; while(x){int y; y=x; print(x+toInt(a));}}`

³Attention, en `pn1` la conditionnelle n'est jamais une expression.