Notions de base sur l'interprétation des langages

Exercice 1

On considère un langage dont les conventions lexicales sont les suivantes :

- les constantes entières signées sont des unités lexicales, ce sont des suites non vides de chiffres (caractère de `0` à `9`) qui peuvent être (ou non) préfixées par le caractère `-` ;
- les identificateurs (noms de variables) sont des unités lexicales, ce sont des suites non vides de lettres alphabétiques majuscules ou minuscules ou de chiffres qui commencent par une lettre alphabétique ;
- les suites de caractères `:=`, `;`, `+` et `*` sont des unités lexicales ;
- les espaces sont des séparateurs.

Les phrases suivantes sont-elles lexicalement bien formées pour ce langage (suite d'unités lexicales) ? Si oui, donner leur découpage en unités lexicales ; si non, dire sur quel caractère se situe l'erreur lexicale.

1. `x:=-3;`
2. `x := -3 ;`
3. `x : = - 3 ;`
4. `x := - 3 ;`
5. `x := 3*y;`
6. `x*3 := y;`
7. `x3 := y-4;`
8. `x 3 := y - 4;`
9. `x 3 := y -4;`
10. `3 := x`
11. `3 =: x`
12. `;;xx333-4444:=5y+6`

Exercice 2

En Java comme en C++, il y a deux façons de commenter du code :

- La séquence `//` permet d'insérer un commentaire sur une seule ligne, qui se termine donc à la fin de la ligne.
- La séquence `/*` permet d'insérer un commentaire sur plusieurs lignes. La séquence `*/` marque la fin du commentaire. En d'autres termes, tout ce qui est situé entre `/*` et `*/` est considéré comme faisant partie du commentaire.

En Ocaml, les commentaires sont introduits par la suite de caractères `(*` (sans espace) et terminés par les caractères `*)`. Les commentaires peuvent être imbriqués. Le commentaire ne se termine donc pas forcément à la première apparition des caractères `*)`.

1. Que se passe-t-il en Java si on met en commentaire du code qui contient déjà un commentaire `/*...*/`
2. Pourquoi l'approche proposée dans Ocaml n'est pas plus généralement répandue ?
3. Un commentaire Java commence parfois par la suite de caractères `/**`. Pourquoi ? Est-ce une convention de commentaires Java différente ?

Exercice 3

On s'intéresse à décrire les différentes étapes de la compilation pour le petit programme python suivant.

```
1  def somme(T):
2      S = 0
3      for i in range(len(T)):
4          S = S + T[i]
5      return(S)
6  # On observe que la somme des entiers de 0 à n-1 est égale à n(n-1)/2
7  n = 10
8  T = list(range(n))
9  if somme(T) == n*(n-1)//2 : print("ok")
10 else: print("ko")
```

On reprend les spécifications d'unités lexicales, règles de syntaxe pour le langage mini-python données en cours et rappelées à la fin de l'exercice.

1. Donner la liste des premières unités lexicales de ce programme correspondant à la définition de la fonction `somme`. Annoter les unités `ID` et `CST` par leur valeur.
2. Donner l'arbre de dérivation syntaxique associé à l'instruction if/then/else du programme précédent. On commencera par donner la suite des unités lexicales correspondantes.

   On s'inspirera de la règle de grammaire pour la boucle **for** pour proposer une règle de grammaire pour l'instruction conditionnelle.
3. Les règles de syntaxe permettent d'interpréter l'expression `n*(n-1)//2` de deux manières différentes. Donner les arbres de dérivation correspondant aux deux choix possibles. Est-ce que le choix de l'une ou l'autre des interprétations a une conséquence sur le résultat ? Quel choix est réalisé par l'interpète Python ? Que faut-il écrire pour obtenir la seconde option ?
4. Compléter la structure d'arbre de syntaxe abstraite vue en cours (quels nouveaux nœuds faut-il introduire ?) pour représenter la structure de ce programme (uniquement la conditionnelle).

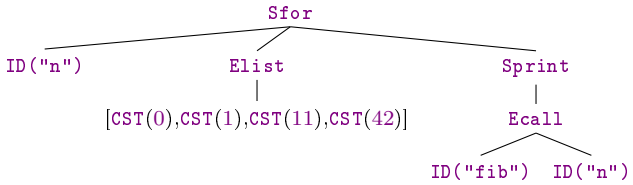
Rappel des unités lexicales

- mots-clés : **def if else return print for in and or not**
- opérations arithmétiques : `PLUS MINUS TIMES DIV MOD`
- comparaisons : `CMP` (une seule unité, associée à l'opérateur de comparaison :`'=='`, `!=`, `>=` ...)
- symboles associés à l'indentation `BEGIN END NL EOF`
- autres symboles spéciaux : `'(' ')' '[' ']' ',' '=' ':'`
- constantes : `CST` (associé à une valeur entière, flottante ou chaîne de caractères)
- identifiants : `ID` associé à une représentation de l'identifiant (chaîne)

Rappel des règles de grammaire

- *expr* ≔ `ID` | `CST` | *unop expr* | *expr binop expr* | `'('` *expr* `')'`
  | `ID '('` *lexpr* `')'` | *expr* `'['` *expr* `']'` | `'['` *lexpr* `']'`
- *instr* ≔ **for** `ID` **in** *expr* `':'` *suiteinstr*
- *suiteinstr* ≔ *simplinstr* `NL` | `NL BEGIN` *linstr* `END`
- *simplinstr* ≔ **print** `'('` *expr* `')'`

Exemple d'arbre de syntaxe abstraite



Exercice 4

On indique pour chacun des programmes Java suivants le message d'erreur renvoyé par le compilateur `javac`. Expliquer à quelle étape est détectée chaque erreur.

```
1  class A1 {
2      int x?;
3      public static void main(){
4      }
5  }
6  class A2 {
7      static int m(int x){
8          return (((4+x) * x/3 +1);
9      }
10     public static void main(){
11         m(42);
12     }
13 }
14 class A3 {
15   int x;
16   public static void main(){
17       String s = "bonjour;
18 ␣␣␣␣␣␣System.out.println(s);
19 ␣␣}
20 }
```

```
ex1.java:2: error: ';' expected
    int x?;
         ^
ex1.java:8: error: ')' expected
        return (((4+x) * x/3 +1);
                                ^
ex1.java:17: error: unclosed string literal
      String s = "bonjour;
                 ^
```

```
1  class A4 {
2      public static void main(){
3          int y=3;
4          System.out.println(x+y);
5      }
6  }
7
8  class A5 {
9      static void m(int x){
10         System.out.println(x+1);
11     }
```

```
12      public static void main(){
13          int y=3;
14          m(y=y+4);
15          //          m(y==y+4);
16      }
17  }
```

```
ex2.java:4: error: cannot find symbol
        System.out.println(x+y);
                           ^
  symbol:   variable x
  location: class A3
ex2.java:15: error: incompatible types: boolean cannot be converted to int
        m(y==y+4);
          ^
```

```
1   class A6 {
2       static int m(int x){
3           if (x>0) return 1;
4           else
5               if (x<=0) return 2;
6               else return 3;
7           return 4;
8       }
9       public static void main(){
10          m(42);
11      }
12  }
```

```
ex3.java:7: error: unreachable statement
        return 4;
        ^
```

```
1   class A7 {
2       static int m(int x){
3           return (x+1);
4       }
5       public static void main(){
6           int final=42;
7           m(final+1);
8       }
9   }
```

```
ex4.java:6: error: not a statement
        int final=42;
        ^
ex4.java:6: error: ';' expected
        int final=42;
           ^
ex4.java:6: error: illegal start of type
```

```
        int final=42;
                ^
ex4.java:7: error: illegal start of expression
        m(final+1);
          ^
ex4.java:7: error: illegal start of type
        m(final+1);
               ^
```

```
1  class A8 {
2      static int m(int x, boolean b, double f){
3          return b?x+(int)f:3;
4      }
5      public static void main(){
6          m(4,3.14);
7      }
8  }
9  class A9 {
10     static int m(int x, boolean b, double f){
11         return b?x+(int)f:3;
12     }
13     public static void main(){
14         m(4.0,true,3.14);
15     }
16 }
```

```
x5.java:6: error: method m in class A8 cannot be applied to given types;
        m(4,3.14);
        ^
  required: int,boolean,double
  found: int,double
  reason: actual and formal argument lists differ in length
ex5.java:14: error: incompatible types: possible lossy conversion from double to int
        m(4.0,true,3.14);
          ^
```

```
1  class A10 {
2      public static void main(){
3          short s = 35000;
4          System.out.println(s);
5      }
6  }
7  class A11 {
8      static int m(int x){
9          final int y=x+1;
10         y++;
11         return y;
12     }
13     public static void main(){
14         m(4);
15     }
16 }
```

```
ex6.java:3: error: incompatible types: possible lossy conversion from int to short
        short s = 35000;
                  ^
ex6.java:10: error: cannot assign a value to final variable y
        y++;
        ^
```

```java
class A {
    public static void main(String args[]){
        int t[]=new int[2];
        t[2]=42;
        System.out.println(t[2]);
    }
}
```

Que se passe-t-il à la compilation et à l'exécution de ce programme ?