# Implémentation d'un compilateur simplifié

Federico Pfeiffer
Juin 2017

# Vue d'ensemble

1) État général de l'implémentation
2) Utilisation
3) Processus général de la compilation
4) Vue d'ensemble de l'architecture
5) Construction arbre abstrait / TDS
6) Vérification sémantique
7) Production du code
8) Exemples d’exécution

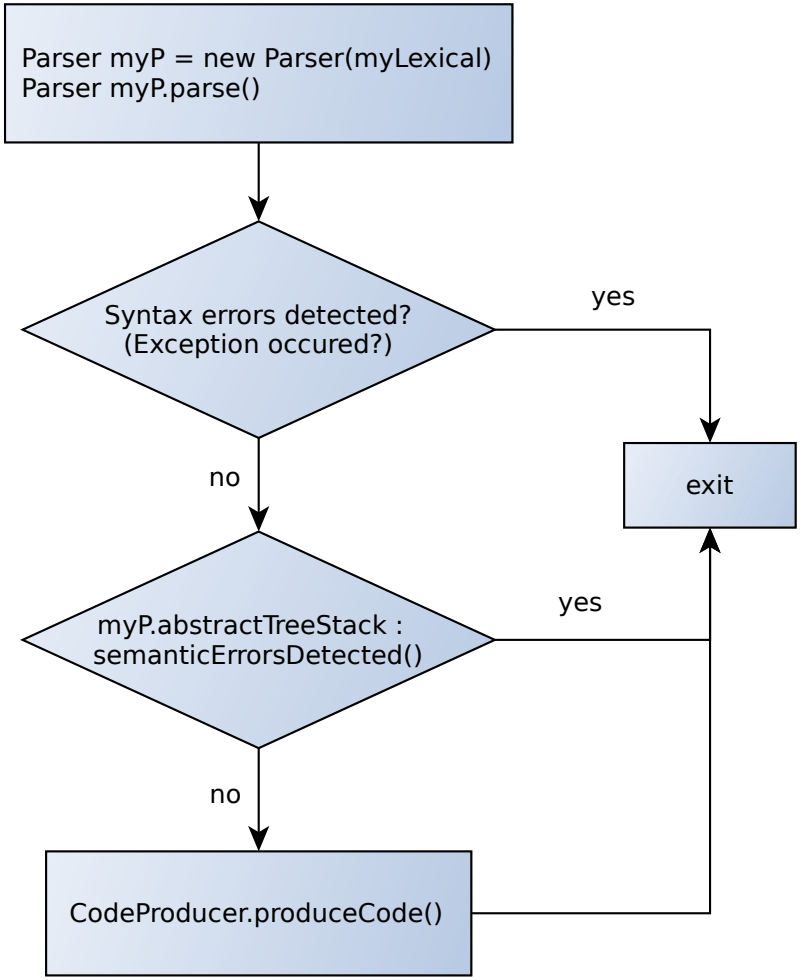
# État général de l'implémentation

- Implémtenté selon le cahier des charges demandé
- Récursivité
- Portée des variables
- Tableaux

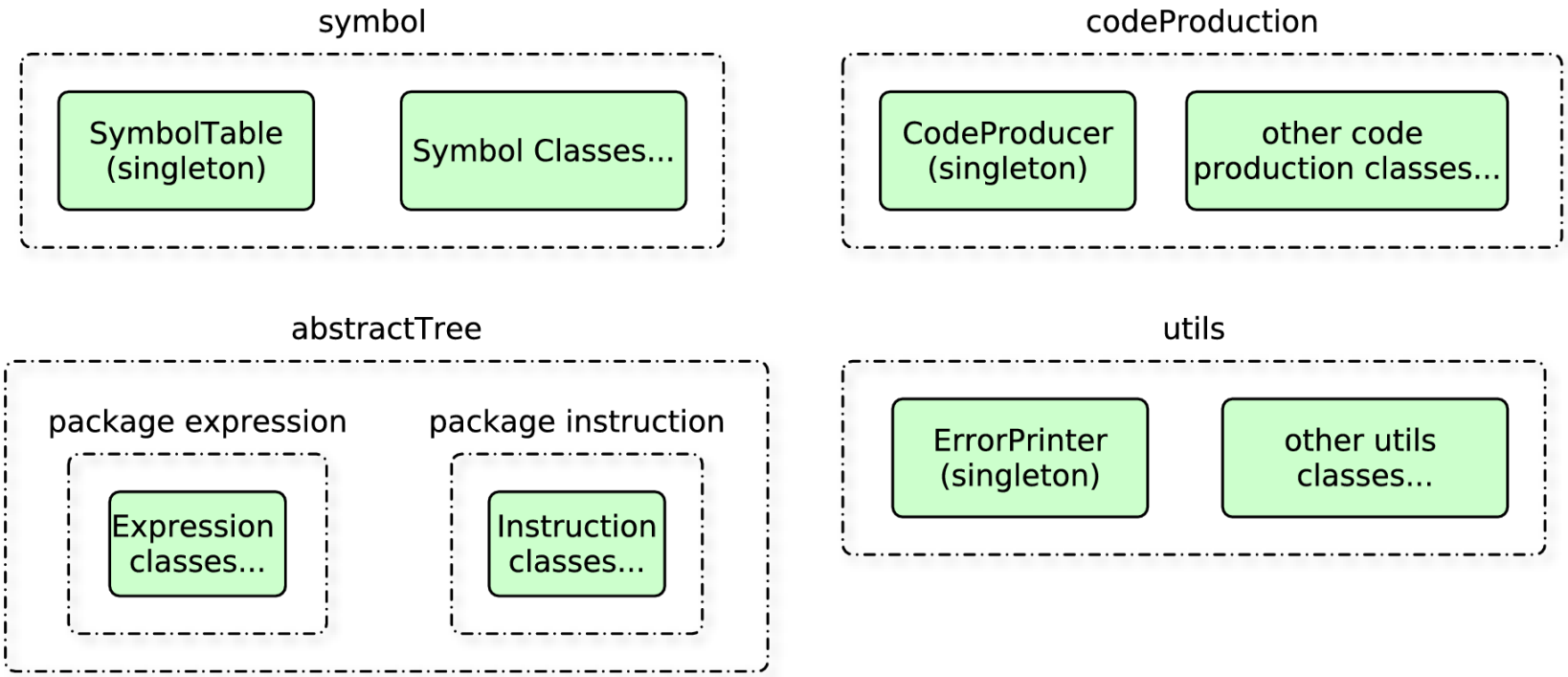
# Utilisation

```
# compiler le compilateur
$ make

# compiler le fichier hepial
$ bash hepiaCompile <fileName >  # fileName = input .txt par défaut

# lancer l exécutable compilé
$ bash <programName >
```

# Processus de compilation



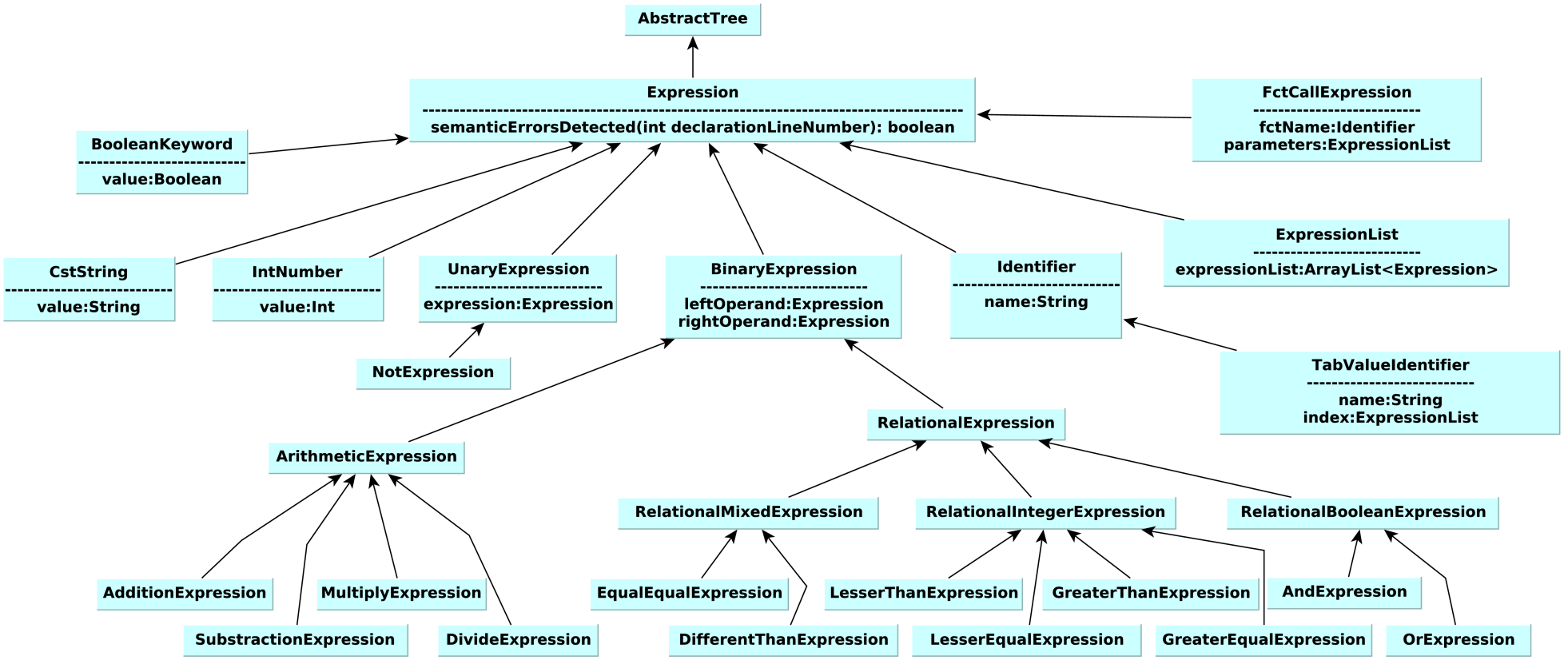
# Vue d'ensemble des packages



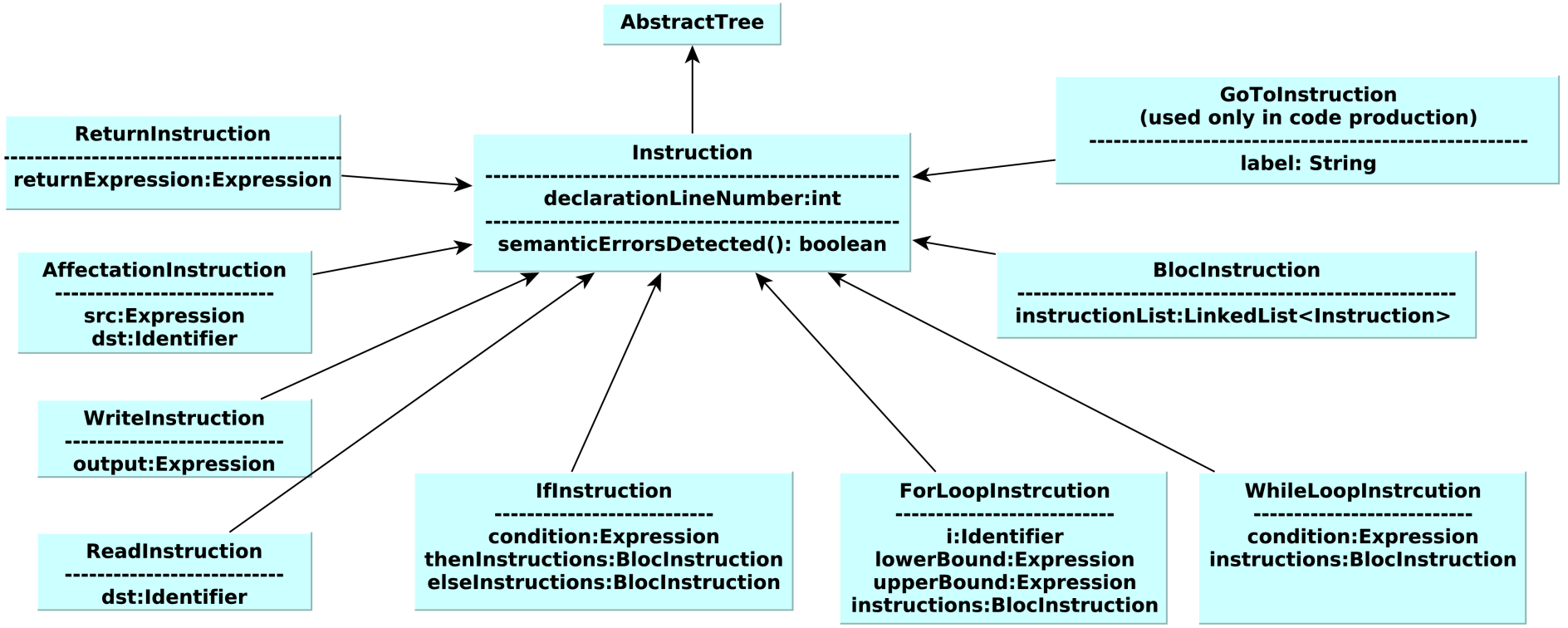
# Construction arbre abstrait / TDS

- Générés pendant le parsing CUP
- Éléments de la pile de l'arbre : asbtractTree
- A la fin du parsing : un seul élément dans la pile

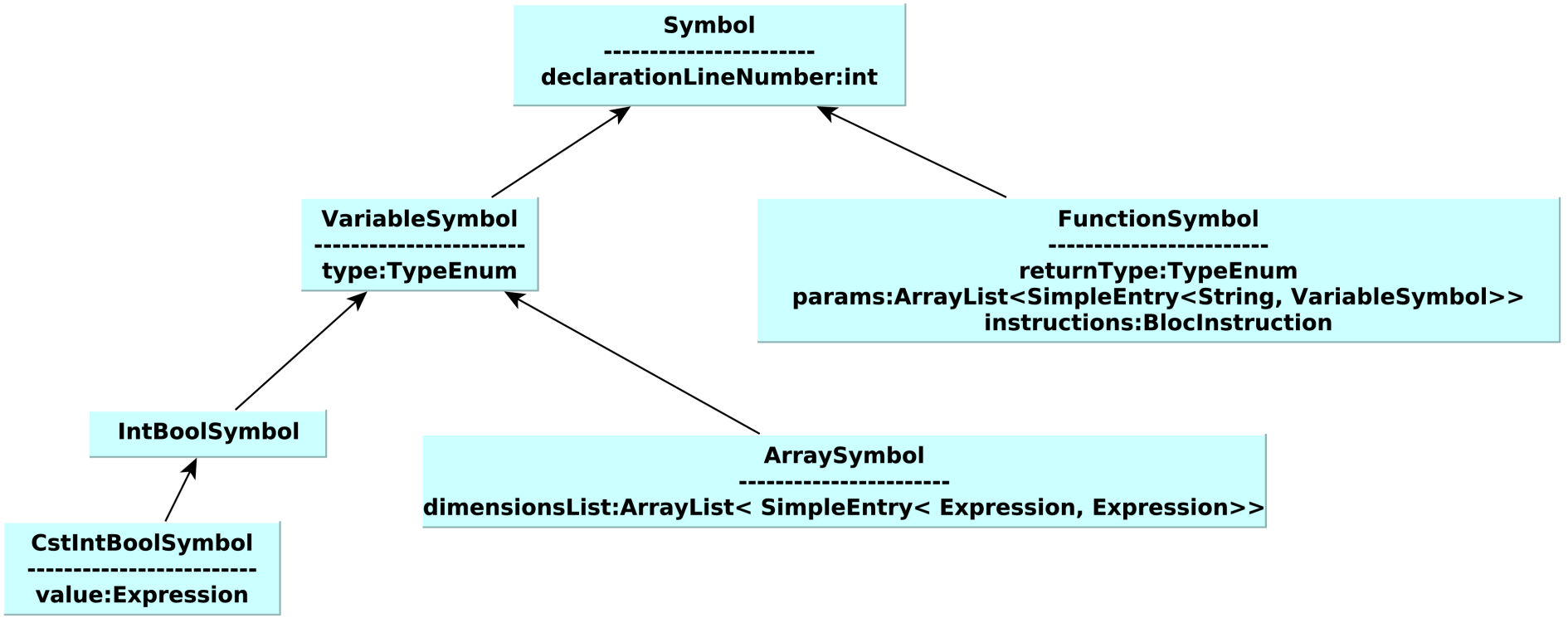
# Construction arbre abstrait / TDS (2)



# Construction arbre abstrait / TDS (3)



# Construction arbre abstrait / TDS (4)



# Vérification sémantique

- Pattern utilisé : « pattern controller »

```
// check for semantic errors and print errors if any
boolean errorsDetected = false ;
if( SymbolTable . getInstance (). semanticErrorsDetected ()){
errorsDetected = true ;
}
if( abstractTreeElement . semanticErrorsDetected ()){
errorsDetected = true ;
}
if( errorsDetected ){
ErrorPrinter . getInstance (). printErrors() ;
System .exit(1) ;
}
```

# Affichage des erreurs

- Singleton : ErrorPrinter
- ErrorPrinter.logErrors()
- ErrprPrinter.printErrors()

# Production du code

```
public class HepialProgramName {
  public static void main ( String [] arg ){
    MainBlock mainBlock = new MainBlock() ;
    mainBlock . mainFunction() ;
  }
}

public class MainBlock {
  // Hepial variables defined in main block go here

  public void mainFunction (){
    // Hepial main instructions go here
  }
}
```

# Production du code (2)

Enjeux :

- Moyen simple de débugger jasmin
- Portée des variables → attributs au lieu de .locals
- Récursivité

# Production du code (3)

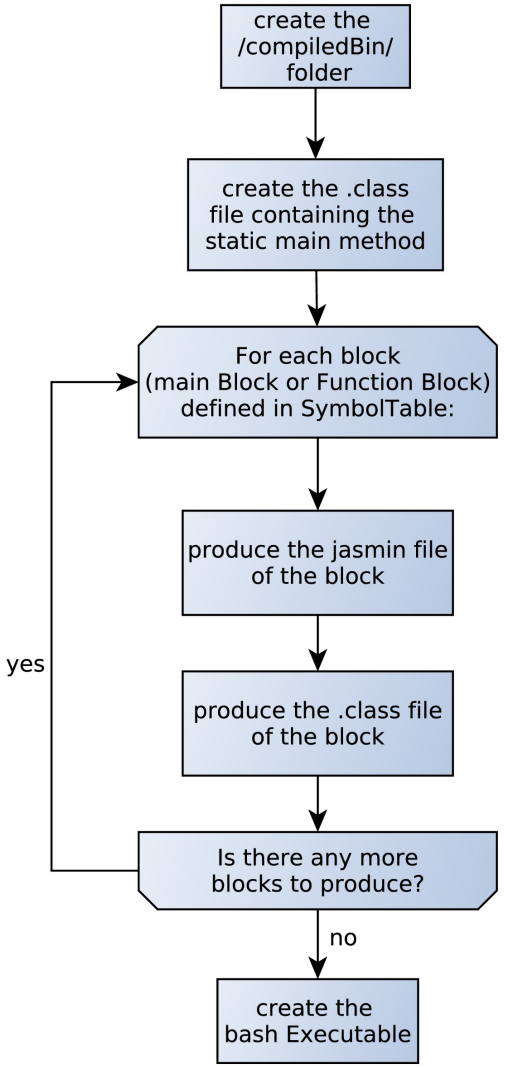
## Architecture d'une fonction:

```
programme hepialProgramName

constante entier globale = 1 ;
entier resultat ;

entier maFct ( entier param1 )
    entier locale ;
    debutfonc
        locale = param1 + globale ;
        retourne locale ;
    finfonc

debutprg
    resultat = maFct(2) ;
finprg
```

# Production du code (4)

Vue d'ensemble :

CodeProducer.produceCode()



# Exemple d’exécution

```
1 :  programme hepialProgramName
2 :
3 : constante entier const = 1 ;
4 : booleen bool ;
5 : entier bool ;
6 :
7 : debutprg
8 :        const = 123 ;
9 :        bool = bool + 321 ;
10 :       bool = indefini ;
11 :finprg
```

# Exemple d’exécution (2)

```
programme hepialProgramName
    constante booleen true = vrai ;
    constante booleen false = faux ;
debutprg
    si true ou false alors
        ecrire "OK1" ;
    sinon
        ecrire "KO" ;
    finsi
    si true et false alors
        ecrire "KO" ;
    sinon
        ecrire "OK2" ;
    finsi
    si true ou false et 1 < 2 alors
        ecrire "OK3" ;
    sinon
        ecrire "KO" ;
    finsi
    si true et false ou 1 < 2 alors
        ecrire "OK4" ;
    sinon
        ecrire "KO" ;
    finsi
finprg
```

# Exemple d’exécution (3)

```
programme hepialProgramName
    entier globale ;
    entier memeNom ;
    entier fct ()
        entier memeNom ;
        debutfonc
            ecrire " globale depuis fct() :" ;
            ecrire globale ;
            memeNom = 3 ;
            ecrire " memeNom depuis fct() :" ;
            ecrire memeNom ;
            retourne memeNom ;
        finfonc
    debutprg
        globale = 1 ;
        memeNom = 2 ;
        ecrire " memeNom dans main :" ;
        ecrire memeNom ;
        memeNom = fct() ;
        ecrire " memeNom dans main après fct () call:" ;
        ecrire memeNom ;
    finprg
```

# Exemple d’exécution (4)

```
programme HepialProgramName
    constante entier n = 4 ;
    entier result ;
    entier facto ( entier n)
        entier asdf ; // déclaration ici sinon la grammaire ne fonctionne pas.
        debutfonc
            si n == 0 alors
                retourne (1) ;
            sinon
                retourne (n* facto (n-1)) ;
            finsi
        finfonc
    debutprg
        si n <20 alors
            result = facto (n) ;
        sinon
            ecrire " votre nombre est trop grand ! " ;
        finsi
        ecrire " factorielle de ";
        ecrire n ;
        ecrire " est égale à " ;
        ecrire result ;
    finprg
```