4A IR - CS410

TP Languages et Compilation
2013 / 2014

Introduction

- Bassem Debbabi (TP1)
- Mouna Tka (TP2)

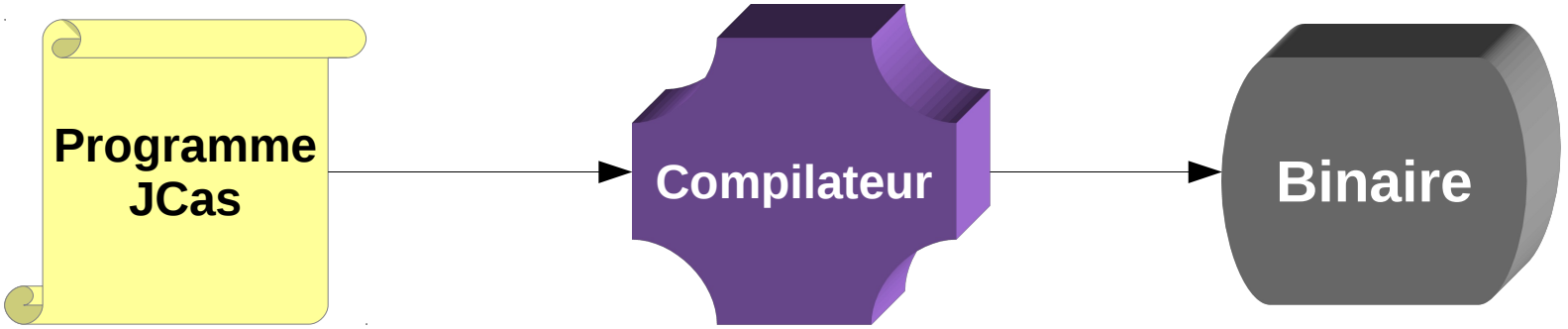
Plan

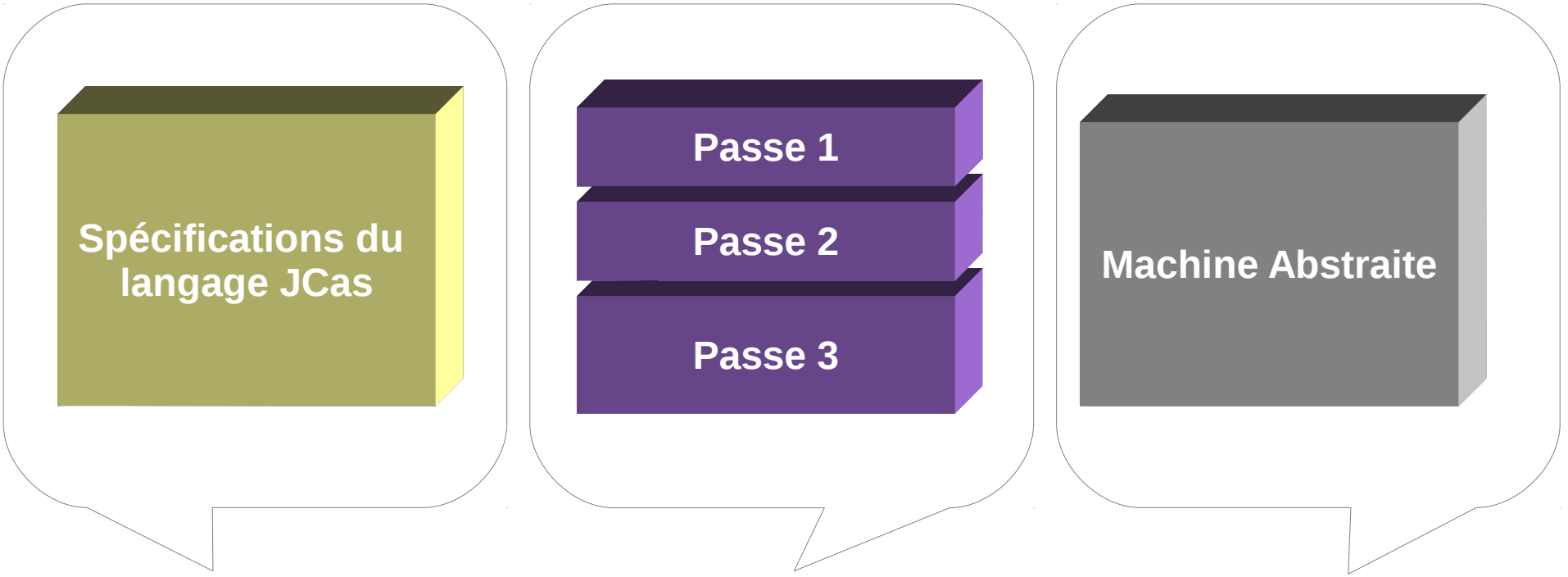
- Présentation du projet
- Le langage JCas
- Vue d'ensemble du compilateur
- Environnement de développement
- Planning
- Références

Présentation du projet

- But du projet :
  - écrire un compilateur “zéro-défaut” pour le langage JCas;
  - utiliser des générateurs d'analyseurs lexical et syntaxique (JFlex et Cup);
  - écrire des séries de tests unitaires;
  - travailler en équipe.

Présentation du projet





Fournies

Partiellement fournies
A compléter!

Intérpréteur Fourni

Le langage JCas

- Langage qui ressemble à Pascal ou Ada, sans fonctions ni procédures.
- Example de programme JCas :

```
-- Calcul de la factorielle
program
     n, fact : integer ;
begin
     write(“Entrer un entier : “) ;
     read(n) ;
     fact := 1 ;
     while n >= 1 do
          fact := fact * n ;
          n := n – 1 ;
     end ;
     write(“fact(“, n, “) = “, fact) ;
     new_line ;
end.
```

Le langage JCas

- Spécification du langage :

  a) *Lexicographie*

  - Voir `Lexicographie.txt` pages 2
  - La lexicographie définit les mots (ou lexèmes) du langage JCas.

Exercice:

Les chaînes suivantes sont-elles des identificateurs du langage JCas?

`toto, toto_1, toto__1, 2_a, _toto`

Les chaînes suivantes sont-elles des constantes entières du langage JCas?

`12, -12, 12e2, 12.5e2, 12e+2`

Les chaines suivantes sont-elles des constantes réelles du langage JCas?

`0.12, .12, 1.5e+3, 1.5e-3, 1e-2, 12, 1.2e++2`

Le langage JCas

- Spécification du langage (*suite*) :

  b) *Syntaxe hors-contexte*

  - Voir `Syntaxe.txt` page 4
  - La syntaxe hors-contexte définit les phrases du langage JCas.

Exercice:

Écrire un programe JCas qui ne fait rien.

Le langage JCas

- Spécification du langage (*suite*) :

  c) Syntaxe contextuelle

  - Voir `Context.txt` page 6
  - La syntaxe contextuelle (ou sémantique statique) du langage JCas définit:
    - les règles de déclaration des identificateurs ;
    - les règles d'utilisation des identificateurs ;
    - les règles de typage des expressions.

Exercice:

Faire la liste de tous les messages d'erreurs contextuelles. Pour chaque message d'erreur, donner un exemple de prgramme JCas.

Vue d'ensemble du compilateur

- Le compilateur JCas comporte trois passes (le programme va être parcouru trois fois).
  - **Passe 1**
    - Analyse lexicale et syntaxique
  - **Passe 2**
    - Vérifications contextuelles et décoration de l'arbre abstrait
  - **Passe 3**
    - Génération de code
- Cela permet de bien décomposer les problèmes.

Vue d'ensemble du compilateur – Passe 1

- **Analyse lexicale**
  - Consiste à décomposer un programme JCas, donné sous la forme d'une suite de caractères, en une suite de mots (ou lexèmes).
  - A chaque unité lexicale reconnue est associée une *unité lexicale* (ou «*symbole*»).
  - Les différentes unités lexicales sont définies dans le fichier `sym.java` généré automatiquement par Cup.

Exemple:

La suite de caractères

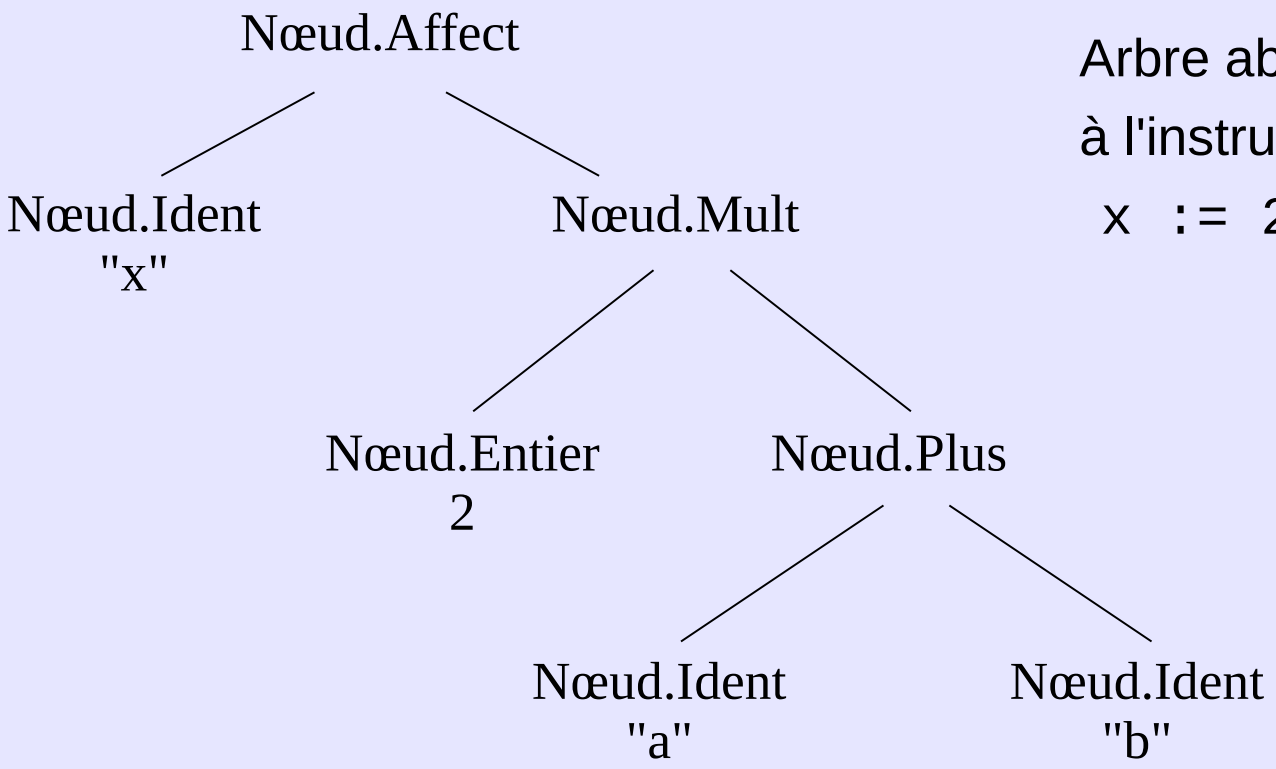
```
x := 2 * (a + b) ;
```

correspond à la suite d'unités lexicales :

```
IDF(x)
AFFECT
CONST_ENT(2)
MULT
PAR_OUVR
IDF(a)
PLUS
IDF(b)
PAR_FERM
POINT_VIRGULE
```

Vue d'ensemble du compilateur – Passe 1

- **Analyse syntaxique**
  - Consiste à déterminer si une suite de mots est une phrase du langage et à construire un arbre abstrait du programme.



Exemple:

Arbre abstrait correspondant à l'instruction

```
x := 2 * (a + b) ;
```

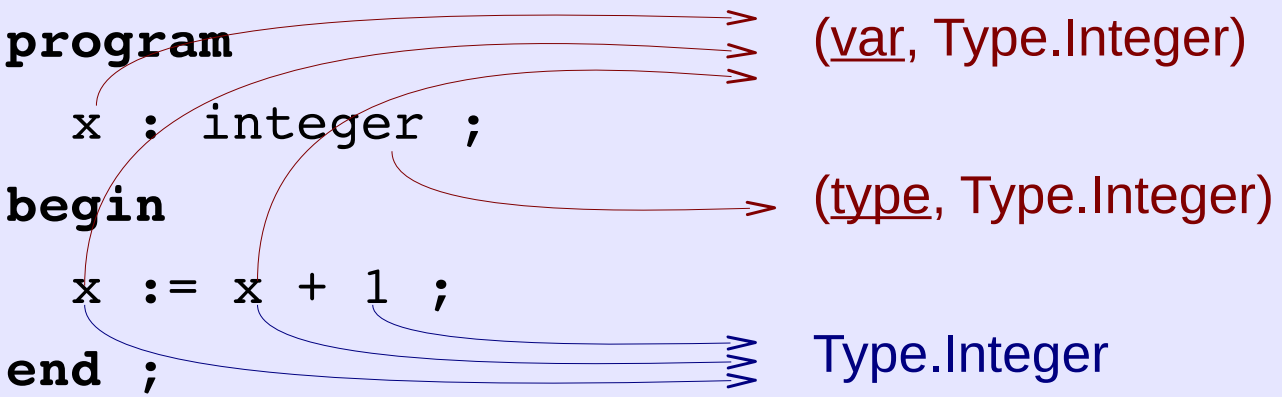
Vue d'ensemble du compilateur – Passe 2

- Vérifications contextuelles et décoration de l'arbre abstrait
  - vérifier qu'un programme JCas est contextuellement correct;
  - décorer l'arbre abstrait du programme.
- Principe :
  - On construit un *environnement*, qui associe à tout *identificateur* sa *définition*.
  - Une *définition* est un couple (nature, type).
  - Dans le langage JCas, on distingue les natures d'identificateurs suivantes :
    - Les identificateurs de types ;
    - Les identificateurs de constrantes ;
    - Les identificateurs de variables.

Vue d'ensemble du compilateur – Passe 2

- Principe (*suite*):
  - Les identificateurs de types et de constantes sont uniquement des identificateurs prédéfinis (on ne peut déclarer que des identificateurs de variable).
  - On décore les identificateurs avec leur définition et les expressions avec leur type.

Exemple:



Vue d'ensemble du compilateur – Passe 2

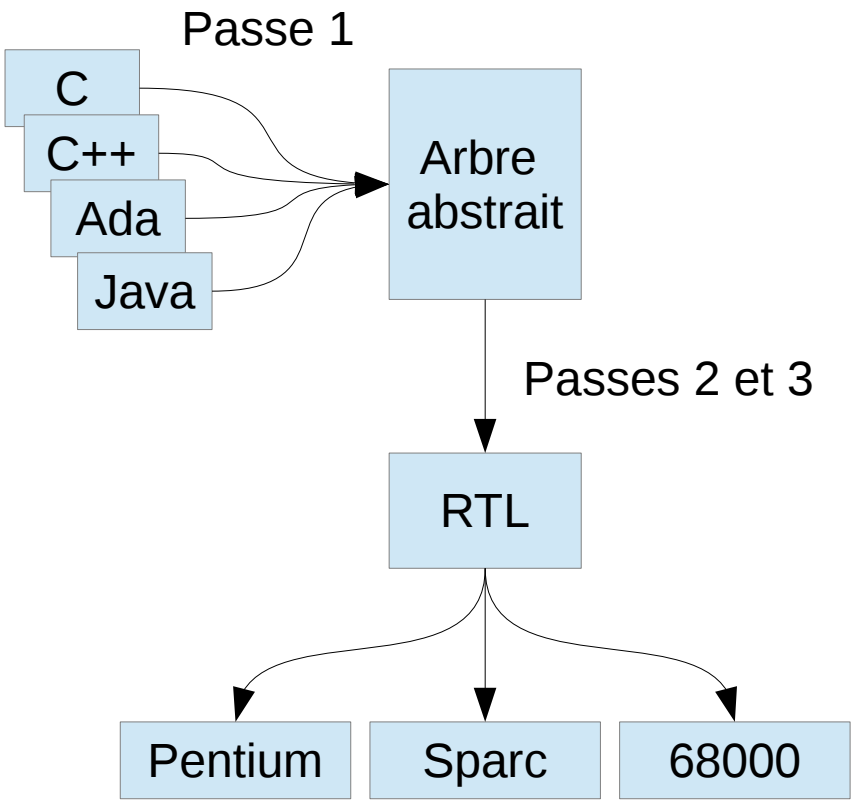
- Vérification contextuelles
  - Les vérifications contextuelles sont réalisées par un parcours de l'arbre abstrait du programme.
  - Le parcours des *déclarations* permet de construire l'environnement.
  - Le parcours des *instructions* permet de vérifier que les identificateurs sont utilisés conformément à leur déclaration, et que les expressions sont bien typées.
  - Lors de ce parcours, l'arbre abstrait est décoré, afin de préparer la passe 3.

Vue d'ensemble du compilateur – Passe 3

- La passe 3 consiste à parcourir l'arbre abstrait décoré une seconde fois et à produire du code exécutable.
  - On produit du code pour une machine abstraite proche du 68000 (voir `Machine_Abstraite.txt` page 15)
  - Intérêts d'utiliser une machine abstraite :
    - Faire abstraction des particularités de bas niveau des langages assembleurs (comme par exemple les problèmes d'alignement en 68000) ;
    - permettre la production de code assembleur réel pour plusieurs machines similaires
      - écrire facilement plusieurs “back-ends” de compilateurs)

Vue d'ensemble du compilateur – Passe 3

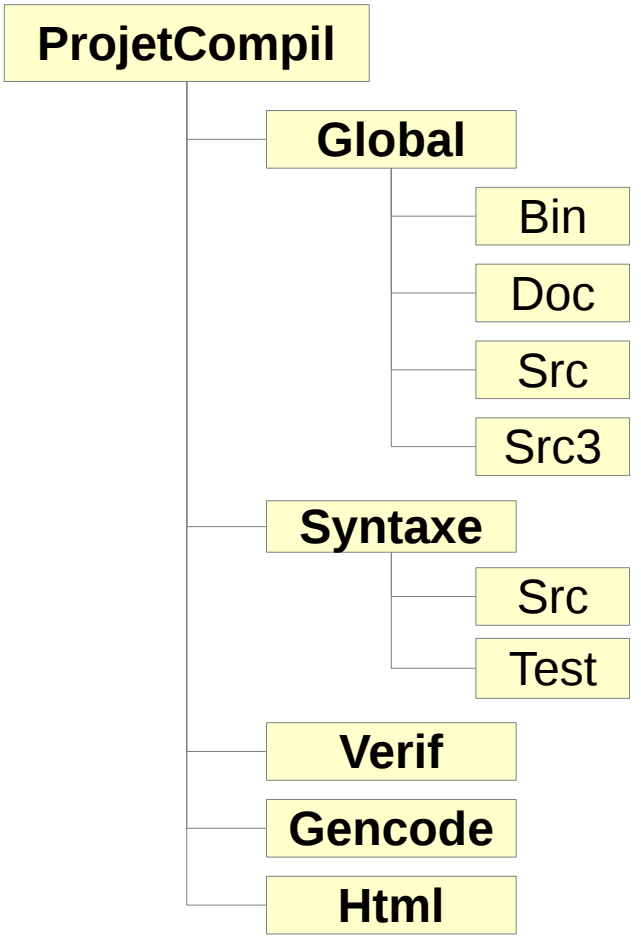
- *Exemple de gcc*
  - Le compilateur gcc peut compiler des programmes écrits en C, C++, Ada ou Java.
  - Gcc utilise une structure d'arbre unique pour tous ces langages.
  - Gcc produit du code “RTL” (Register Transfer Language), code pour une machine abstraite dont la syntaxe est proche du Lisp.
  - Il y a plusieurs “back-ends” pour différentes machines.



Environnement de programmation

- Le projet est développé
  - en binômes (pour les passes 1 et 2) ;
  - en equipe (pour la passe 3).
- Récupération du projet :
  - http://esisar.net/cs410/ProjetCompil.tar.gz
    - Utilisateur: **esisar**
    - Mot de passe: **jcas**
  - Extraire le fichier compressé dans votre "***home-directory***"

- Organisation des répertoires



Environnement de programmation

- Commandes du projet
  - Variable d'environnement

```
$ export CLASSPATH=$CLASSPATH:$HOME:
$HOME/ProjetCompil/Global/Bin/java-cup-11a-runtime.ja
r:.
```

  - Compilation
    - Dans le répertoire `ProjetCompil/Syntaxe/Src` ou `ProjetCompil/Verif/Src` :

```
$ make

$ make clean
```

    - (à ne pas faire avant chaque compilation !)

Planning

Séance	À faire	À rendre
1	Présentation du projet + début pass 1	
2	Fin passe 1	
3-5	Passe 2	
6	Tests passe 2 (Cobertura)	Passes 1 et 2 à rendre le 25 Oct. avant 18h00.
7-11	Passe 3	Passe 3 à rendre le 06 Déc. avant 18h00.

References

- Page du projet
  - http://esisar.net/cs410
- JFlex
  - http://jflex.de/manual.html
- Cup
  - http://www2.cs.tum.edu/projects/cup/manual.html
- Cours de compilation
  - http://membres-liglab.imag.fr/oriat/Compil/