Projet de Compilation

Projet de compilation du langage BLOOD
-1er Rapport-

mask mask
moi moi
mask mask
mask mask

Année 2020-2021

Encadrants :
Suzanne Colin
Sébastien DA SILVA
Gérald OSTER

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : mask, mask

Élève-ingénieur(e) régulièrement inscrit(e) en 2e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 31707457

Année universitaire : 2020-2021

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

Projet de compilation du langage BLOOD

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Villers-lès-Nancy, le 10 novembre 2021

Signature :

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : moi, moi

Élève-ingénieur(e) régulièrement inscrit(e) en 2e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 31917080

Année universitaire : 2020-2021

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

Projet de compilation du langage BLOOD

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Villers-lès-Nancy, le 10 novembre 2021

Signature :

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : mask, mask

Élève-ingénieur(e) régulièrement inscrit(e) en 2e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 31733737

Année universitaire : 2020-2021

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

Projet de compilation du langage BLOOD

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Villers-lès-Nancy, le 10 novembre 2021

Signature :

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : mask, mask

Élève-ingénieur(e) régulièrement inscrit(e) en 2e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 31706721

Année universitaire : 2020-2021

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

Projet de compilation du langage BLOOD

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Villers-lès-Nancy, le 10 novembre 2021

Signature :

Projet de Compilation

Projet de compilation du langage BLOOD
-1er Rapport-

mask mask
moi moi
mask mask
mask mask

Année 2020-2021

mask mask
moi moi
mask mask
mask mask
mask.mask@telecomnancy.eu
moi.moi@telecomnancy.eu
mask.mask@telecomnancy.eu
mask.mask@telecomnancy.eu

TELECOM Nancy
193 avenue Paul Muller,
CS 90172, VILLERS-LÈS-NANCY
+33 (0)3 83 68 26 00
contact@telecomnancy.eu

TELECOM Nancy
193, avenue Paul Muller,
CS 90172, VILLERS-LÈS-NANCY
+33 (0)3 83 68 26 00

Encadrant :
Suzanne Colin
Sébastien DA SILVA
Gérald OSTER

Table des matières

Table des matières **vi**

1 Fiche projet **ix**

1.1 Auteurs ix

1.2 Cadrage ix

1.2.1 Finalités et importance du projet dans le cursus ix

1.2.2 Objectifs et résultats opérationnels x

1.3 Déroulement du projet x

2 Introduction **xiv**

2.1 Historique xiv

2.2 Le projet xiv

3 Grammaire du langage **xvi**

3.1 Spécifications xvi

3.2 LL(1) xvi

3.3 Difficultés rencontrées xvi

4 Structure de l'arbre abstrait **xvii**

4.1 Choix effectués xvii

4.2 Noeuds de l'AST xvii

5 Jeux d'essais **xxi**

5.1 Définition des classes xxi

5.2 Structure conditionnelle et structure de boucle xxiii

5.3 Opérateurs arithmétiques et concaténation de chaînes de caractères . . . xxiv

5.4 Messages xxvi

5.5 Bloc principal et instanciation xxvi

6 Conclusion et Bilan **xxviii**

6.1 Conclusion xxviii

6.2 Bilan des membres de l'équipe projet xxix

6.2.1 Bilan individuel d'mask mask xxix

6.2.2 Bilan individuel de moi moi xxx

6.2.3 Bilan individuel de mask mask xxxi

6.2.4 Bilan individuel de mask mask xxxi

6.3 Travail réalisé xxxii

Bibliographie / Webographie **xxxiii**

Annexes **xxxv**

A CR Réunion 12/10/2020 **xxxv**

I Prise de contact, retour sur la perception du sujet (incompréhensions, but recherché), et retour sur le TP1 xxxv

II Choix du chef de projet xxxvi

III Création du dépôt git xxxvi

IV Plannification des premiers objectifs (Grammaire, AST, TDS) . . xxxvi

B CR Réunion 22/10/2020 **xxxviii**

I Retour sur le jalon effectué depuis la dernière réunion : construction de la grammaire, début des tests xxxviii

II Les difficultés rencontrées xxxviii

III Problème d'ambiguïté dans la grammaire xxxix

C CR Réunion 31/10/2020 **xli**

I Retour sur le jalon effectué depuis la dernière réunion xli

II La détection des String xlii

III La détection du bloc comportant 'is' xlii

IV La détection du mot 'return' xlii

V La détection des éléments négatifs xlii

VI La détection des commentaires xliii

VII La détection de 'this' et 'super' xliii
VIII Explication sur la façon de faire un AST avec Antlr et exemples . xliii

D CR Réunion 25/11/2020 **xlv**

I Perfectionnement de l'AST xlviii
II Mise en forme du dépôt pour le rendu (fichiers tests et .MD) . . . xlix

E CR Réunion 06/01/2021 **l**

I Prise en compte des remarques suite à la soutenance l
II Rapport : Affectation des éléments manquants et discussion sur leur contenu liv

1 Fiche projet

1.1 Auteurs

Noms	Qualité
mask mask	Membre
moi moi	Membre
mask mask	Chef de projet
mask mask	Membre
Professeurs	Project Owners

1.2 Cadrage

1.2.1 Finalités et importance du projet dans le cursus

Ce projet a pour but de nous confronter à la conception d'un compilateur en mobilisant les connaissances et compétences acquises dans les modules TRAD1-2 de seconde année à TELECOM Nancy. La mise en œuvre des connaissances acquises en MI1-2 (connaissances théoriques sur les compilateurs descendants, notion de grammaire LL(1), grammaire ambigüe...) nous est également demandée, ceci afin de créer une grammaire correcte et non ambiguë. Par ailleurs, la maîtrise du module de Gestion de Projet de projet est sous-entendue.

1.2.2 Objectifs et résultats opérationnels

Livrable	Contraintes
Code Source	-Respecter les jalons. -Fournir une documentation (conception, notes de développement), installation, compilation, exécution, validation de vos réalisations.
Tests	-Créer et expérimenter un jeu de test varié sur la grammaire obtenue. -Le jeu de tests devra vérifier : - expressions arithmétiques (illustrant priorité et associativité des opérateurs), affectations, déclarations de variables. - if, while, avec des instructions et if / while imbriqués. - définitions de classe, constructeur, méthodes, et paramètres. - appels de méthodes.
Rapport synthétique	Éléments de Gestion de projet (Compte-Rendus de réunions, GANTT, fiche d'évaluation de la répartition du travail, répartition des tâches au sein du groupe).

Table 1.1 – Critères issus du sujet

1.3 Déroulement du projet

Jalons

Numéro	Description
0	Début du projet
1	Grammaire
2	AST
3	Mise en forme du dépôt pour la soutenance
4	Prise en compte des remarques suite à la soutenance

Forces et faiblesses de chacun

Membre	Forces	Faiblesses
moi moi	Maîtrise de LaTeX Connaissance des outils de gestion de projet Capacité de travail	Maîtrise des différents types de grammaires et d'analyses perfectible en sortie de première année Disponibilité limitée pendant les horaires de travail courants -souvent plus disponible tard le soir-
mask mask	Maîtrise de LaTeX Débrouillardise	Difficulté dans la mise en application de certaines méthodes liées à la construction de grammaires
mask mask	Maîtrise de LaTeX Bonne connaissance des cours de MI et de PCL	Vitesse de travail parfois plus lente que celle des autres membres du groupe
mask mask	Bonne compréhension du cours de MI et des grammaires Rapidité dans le travail	Peu à l'aise avec LaTeX

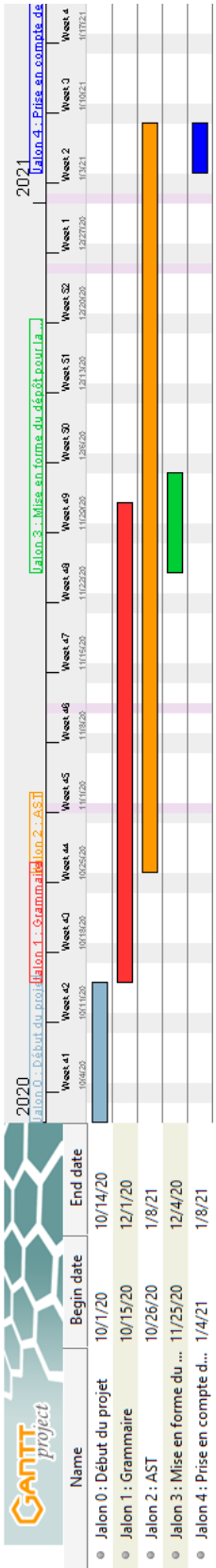
Répartition des tâches au sein du groupe - Matrice RACI - Voir 6.3 pour le temps consacré

Les tâches sont attribuées entre chaque membre du groupe au regard des forces et faiblesses listées précédemment, de la volonté de chacun d'effectuer un jalon particulier et du temps disponible. La répartition des jalons a été réalisée en plusieurs étapes lors du projet en fonction de l'avancement de chacun dans le travail.

Tâche	mask	moi	mask	mask
Jalon 0 Début du projet				
Prise de contact, retour sur la perception du sujet	R	R	R	R
Création du dépôt git	I	I	AR	I
Retour sur le TP1	R	R	I	R
Jalon 1 Grammaire				
Résolution de l'ambiguïté	I	R	I	AR
Complétude	I	I	I	AR
Reconnaissance des classes	I	AR	R	I
Méthodes	AR	I	I	I
Noms de classes et méthodes	I	I	AR	I
Attributs	I	R	I	AR
String	R	AR	I	I
Expressions	I	R	I	AR
Instruction If	I	I	I	AR
Instruction While	AR	I	I	I
Structure de blocs	I	R	AR	R
Opérateurs	I	R	AR	I
Commentaires	I	AR	I	I
Jalon 2 AST				
Découverte de la syntaxe	AR	I	I	I
Variables	AR	I	I	I
Messages	R	I	I	AR
Identificateurs	I	AR	I	I
Opérateurs	I	R	AR	I
Méthodes/Classes	AR	I	I	I
If/While	AR	R	I	I
Jalon 3 Mise en forme du dépôt pour la soutenance				
Simplification de la grammaire	I	R	I	AR
Création d'un jeu de test	R	R	AR	R
Import du fichier de tests fourni par les Project Owners	I	I	AR	I
Jalon 4 Prise en compte des remarques suite à la soutenance				
Etiquette MESSAGE dédoublée	I	I	AR	I
Remonter étiquette MESSAGE	I	I	AR	I
Changement de l'étiquette ' :=' pour VALUE pour la définition d'une variable	AR	I	I	I
Verification de l'étiquette ' :=' dans le cas d'une affectation	AR	I	I	I
Création de l'étiquette NEW	I	I	I	AR
Ajout des etiquettes IF (CONDITION) THEN, ELSE	I	AR	I	I

TABLE 1.2 – R= Responsible, A=Accountable, C= Consulted, I=Informed

Diagramme de GANTT



2 Introduction

Un compilateur est un programme informatique qui traduit l'ensemble du code source d'un projet logiciel en code machine avant son exécution. C'est uniquement après cette traduction que le projet sera exécuté par le processeur qui dispose de toutes les instructions sous forme de code machine avant le début du projet. De cette façon, le processeur a à disposition tous les éléments nécessaires pour exécuter le logiciel en question, traiter les données et générer le résultat.[1]

2.1 Historique

Au début de l'informatique, on programmait directement les ordinateurs en langage machine. Cela s'est vite avéré fastidieux. On a très vite essayé d'utiliser les possibilités de l'informatique pour faciliter le travail de programmation.

Une des innovations a consisté à s'affranchir complètement de la machine en élaborant ce que l'on appelle des langages de haut niveau. Ces langages ont introduit un certain nombre de constructions qui n'existent pas dans le langage de la machine.

Pour les utiliser en pratique, il a fallu construire des programmes qui traduisent des énoncés exprimés dans le langage de haut niveau dont se servent les programmeurs, ce que l'on appelle le langage source, en instructions pour la machine cible. Les programmes effectuant ce genre d'opération s'appellent des compilateurs.[2]

2.2 Le projet

L'objectif de ce projet est d'écrire un compilateur d'un mini langage orienté objet [1].

Ce projet est composé de deux parties :

— Le but de la première partie est d'élaborer la grammaire du langage BLOOD, son AST, ainsi qu'un jeu de tests vérifiant son fonctionnement.
Pour réaliser la première partie de ce projet, l'outil à utiliser est **Antlr**, générateur

1. Mini-langage objet dénommé BLOOD pour *Basic Language Object-Oriented of Doom*

d'analyseurs lexical et syntaxique descendant, interfacé avec le langage Java pour les étapes d'analyse lexicale et syntaxique.

— L'objectif de la deuxième partie consiste à générer le code assembleur au format microPIUP/ASM (dans un fichier).

Ce premier rapport rend ainsi compte du travail effectué dans la partie front-end.

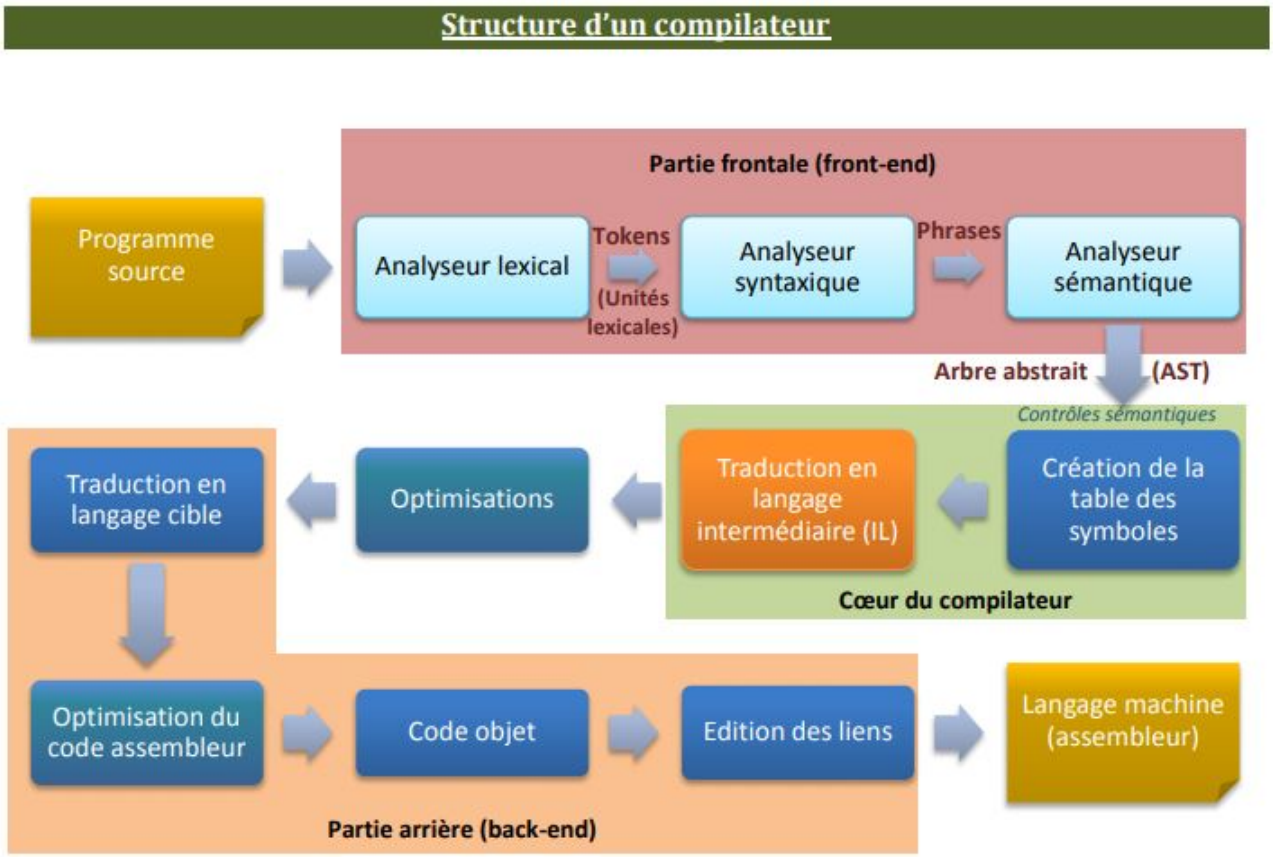


FIGURE 2.1 – Issu du cours de Traduction I, ESIAL 2012

Le compilateur implémenté doit signaler les erreurs lexicales, syntaxiques et sémantiques rencontrées. Lorsqu'une de ces erreurs lexicales ou sémantiques est rencontrée, elle doit être signalée par un message relativement explicite comprenant, dans la mesure du possible, un numéro de ligne. Le compilateur peut s'arrêter après chaque erreur syntaxique détectée (pas d'obligation de reprise). En revanche, le compilateur doit impérativement poursuivre l'analyse après avoir signalé une erreur sémantique.

3 Grammaire du langage

3.1 Spécifications

Le langage BLOOD est un langage inventé orienté objet qui se base sur les spécifications fournies dans un document de Polytech' Paris-Sud. La grammaire ainsi produite ne s'appuie pas sur une grammaire pré-construite, la seule règle à suivre consiste à ce qu'elle soit LL(1).

3.2 LL(1)

L'analyse LL(1) permet de savoir immédiatement quelle règle choisir car seul le premier caractère dicte la règle qui sera suivie. Une grammaire LL(1) permet de ne pas avoir d'ambiguïtés, il n'y aura pas non plus de récursivités droites. Pour nous aider à repérer si une règle n'est pas récursive droite ou plus généralement LL(1) **ANTLRWORKS** intègre un système de détection qui affiche en rouge les règles qui ne sont pas LL(1).

Afin de lever les ambiguïtés de notre grammaire, nous avons, dans un premier temps, dû réécrire la grammaire tout entière afin qu'elle soit LL(1) puis nous nous sommes assurés qu'à chaque nouveau commit il n'y ait pas d'ambiguïtés. En effet, il est très compliqué de lever une ambigüité d'une grammaire qui n'a pas été faite dès le début dans cette optique. De plus, de nombreuses factorisations de règles ont dû être faites pour retirer des récursivités droites et des ambiguïtés.

3.3 Difficultés rencontrées

La plus grande difficulté rencontrée concerne une ligne commençant par un **ID** (un mot quelconque commençant par une minuscule), puisque cela mène à une déclaration de variable ou à des affectations.

Pour régler cela nous avons dû faire de nombreuses factorisations qui rendent la lecture de la grammaire bien plus difficile.

4 Structure de l'arbre abstrait

L'arbre syntaxique abstrait (AST) est généré à la suite de l'analyse syntaxique. Il représente, sous forme d'arbre, le programme, en contenant seulement les éléments importants pour l'analyse sémantique.

4.1 Choix effectués

Afin de simplifier l'arbre abstrait, les éléments qui ne contiennent pas de sémantique ne sont pas présents dans l'AST, comme les virgules ou les parenthèses. De plus, afin de pouvoir sans trop de soucis effectuer l'analyse sémantique, l'ordre des fils est fixe, et leur nom est précisé lorsque c'est nécessaire. Par exemple, le noeud IF a trois fils, qui sont respectivement précédés des noeuds CONDITION, THEN, et ELSE.

4.2 Noeuds de l'AST

Le tableau suivant liste les différents noeuds de l'AST et les noeuds fils associés. Nous avons suivis les règles suivantes :

- Les noms des noeuds sont en majuscules.
- Les noms des noeuds sont en majuscules.
- Les fils sont séparés par des virgules.
- La barre verticale '|' représente le ou exclusif entre plusieurs noeuds.
- L'étoile '*' après un noeud signifie qu'il peut être répété zéro ou plusieurs fois.
- Le plus '+' après un noeud signifie qu'il peut être répété une ou plusieurs fois.
- Le point d'interrogation ' ?' après un noeud signifie qu'il peut être présent zéro ou une fois.

Noeud	Fils
ROOT	DECLARATION
CLASSE	CLASSID , LIST_PARAM ? , ATTRIBUT*, METHOD*,EXTENDS ?
PARAM	ID , CLASSID
BLOCK	ID* , MESSAGE*,VARIABLE_LOCALE*, VARIABLE*, STRING* , AFFECTATION*, SELECT*, IDENTIFICATEUR*, GROUP*
METHOD	((ID ?,CLASSID)|(ID, CLASSID ?), LIST_PARAM ? , (IDENTIFICATEUR | BLOCK | operator))
MESSAGE	((super|this) , (ID|CLASSID|CAST|RESULT|operatorConcat|VARIABLE)) | ((super|this), APPEL_FONCTION) | ((ID|CLASSID|CAST|RESULT|operatorConcat|VARIABLE), APPEL_FONCTION|MESSAGE)
IF	CONDITION , THEN , ELSE
THEN	BLOCK |AFFECTATION | IF | STRING
ELSE	BLOCK | AFFECTATION | IF | STRING
WHILE	CONDITION , DO
DO	BLOCK|AFFECTATION|IF|STRING
STRING	STR , APPEL_FONCTION ?
LIST_PARAM	PARAM* , MESSAGE*, VARIABLE* , IDENTIFICATEUR* , INT* ,STRING*, SELECT* ,(operateurMD|operatorPM)*
PLUS_UNAIRE	INT | VARIABLE
MOINS_UNAIRE	INT | VARIABLE
ATTRIBUT	ID, CLASSID , VALUE ?
VARIABLE	ID | APPEL_FONCTION
VARIABLE_LOCALE	ID , CLASSID, VALUE ?
CONDITION	IDENTIFICATEUR | VARIABLE | operatorComp
AFFECTATION	IDENTIFICATEUR, NEW | INT | operateurMD|operatorPM | operatorComp| operatorConcat| VARIABLE | IDENTIFICATEUR | STRING
SELECT	CLASSID|MESSAGE
IDENTIFICATEUR	MESSAGE | RESULT
APPEL_FONCTION	FONCTION, LIST_PARAM
CAST	CLASSID, CLASSID|IDENTIFICATEUR
GROUP	operatorMD| operatorPM|MESSAGE
FONCTION	ID
NEW	CLASSID, LIST_PARAM
VALUE	INT| operateurMD|operatorPM|NEW
EXTENDS	CLASSID

Table 4.1 – Caption

Le tableau suivant représente les opérations, qui sont représentées par leur symbole :

Noeuds	Fils
operatorLog	
||	STRING| || | MESSAGE | IDENTIFICATEUR , STRING | MESSAGE | IDENTIFICATEUR
&&	STRING | && | MESSAGE | IDENTIFICATEUR , STRING | MESSAGE | IDENTIFICATEUR
operatorConcat	
&	STRING| & | MESSAGE|IDENTIFICATEUR,STRING| & | MESSAGE|IDENTIFICATEUR
operatorComp	
=	INDENTIFICATEUR|MESSAGE|VARIABLE|INT, INDENTIFICATEUR|MESSAGE|VARIABLE|INT
<>	INDENTIFICATEUR|MESSAGE|VARIABLE|INT, INDENTIFICATEUR|MESSAGE|VARIABLE|INT
<	INDENTIFICATEUR|MESSAGE|VARIABLE|INT, INDENTIFICATEUR|MESSAGE|VARIABLE|INT
>	INDENTIFICATEUR|MESSAGE|VARIABLE|INT, INDENTIFICATEUR|MESSAGE|VARIABLE|INT
operatorPM	
+	INT | PLUS_UNAIRE | MOINS_UNAIRE | + | - | / | * | GROUP | VARIABLE | IDENTIFICATEUR , INT | PLUS_UNAIRE | MOINS_UNAIRE | + | - | / | * | GROUP | VARIABLE | IDENTIFICATEUR
-	IINT | PLUS_UNAIRE | MOINS_UNAIRE | + | - | / | * | GROUP | VARIABLE | IDENTIFICATEUR , INT | PLUS_UNAIRE | MOINS_UNAIRE | + | - | / | * | GROUP | VARIABLE | IDENTIFICATEUR
operatorMD	
*	INT | PLUS_UNAIRE | MOINS_UNAIRE | / | * | GROUP | VARIABLE | IDENTIFICATEUR, INT | PLUS_UNAIRE | MOINS_UNAIRE | / | * | GROUP | VARIABLE | IDENTIFICATEUR
/	INT | PLUS_UNAIRE | MOINS_UNAIRE | / | * | GROUP | VARIABLE | IDENTIFICATEUR, INT | PLUS_UNAIRE | MOINS_UNAIRE | / | * | GROUP | VARIABLE | IDENTIFICATEUR

Le tableau suivant représente les noeuds terminaux :

Result
CLASSID
ID
INT
STR
this|super

5 Jeux d'essais

Les jeux d'essais présentés ci-après ont été utilisés pour tester le bon fonctionnement de la grammaire ainsi que pour contrôler la structure de l'AST. Pour ces tests, nous avons créé plusieurs fichiers courts (dans afin de faciliter et d'accélérer les contrôles, chacun se focalisant sur une série de fonctionnalités relativement proches.

5.1 Définition des classes

Le langage BLOOD étant un langage orienté objet, les classes en sont un élément majeur. Pour pouvoir tester l'implémentation, il est nécessaire de tester les éléments consistant une classe, à savoir :

- Le **constructeur** de la classe.
- Les **attributs** (peuvent être définis explicitement dans le corps de la classe ou implicitement dans son en-tête).
- Les **méthodes** (peuvent être définies sous la forme d'une seule expression ou sous la forme d'un bloc).

En plus de ces éléments, il est également nécessaire de tester l'héritage.

Pour tester ces fonctionnalités, nous avons écrit les deux programmes suivants :

```
1  class Ordinateur(var id : Integer) is {
2      var carteGraphique  : CarteGraphique;
3      var processeur      : Processeur;
4
5      var estAllume       : Integer;
6
7      def allumer() is {
8          estAllume := 1;
9      }
10
11     def Ordinateur(var id_constructeur : Integer) is {
12         super(id_constructeur);
13         estAllume := 0;
14         carteGraphique  := Nvidia;
15         processeur := Intel;
16     }
17
```

```
18      def eteindre() is {
19          estAllume := 0;
20      }
21
22      def carteGraphiqueSetter(var carteGraphique : CarteGraphique) is {
23          this.carteGraphique := carteGraphique;
24      }
25  }
26
27  {}
```

Ce premier programme a pour objectif de tester le bon fonctionnement de l'en-tête d'une classe, ainsi que les différentes méthodes de définition d'un attribut.

```
1   class Composant(var id : Integer) is {
2       var estEnPanne : Integer;
3
4       def declencherPanne() is {
5           estEnPanne := 1;
6       }
7
8   }
9
10  class Processeur() extends Composant is {
11      var frequence : Float;
12      var nbCoeur : Integer;
13      var modele : String;
14
15      def Processeur(id : Integer, f : Float, nc : Integer, m : String)
        is {
16          frequence   := f;
17          nbCoeur     := nc;
18          modele      := m;
19      }
20
21  }
22
23  class RAM() extends Composant is {
24      var capacite    : Integer;
25      var type        : String;
26      var static nbBarrettes : Integer;
27
28      def RAM(id : Integer, c : Integer, t : String) is {
29          capacite := c;
30          type := t;
31          nbBarettes := nbBarettes + 1;
32      }
33
34      def static getNbBarrettes() : Integer := nbBarrettes
35  }
36  {}
```

Ce second programme a pour but de tester l'héritage, les différentes méthodes de définition des méthodes et le bon fonctionnement du mot-clé static.

5.2 Structure conditionnelle et structure de boucle

Le langage BLOOD nécessite également l'implémentation des structures conditionnelles `if ... then ... else ...` et du formalisme de la boucle `while ... do ...`. Pour tester ces éléments, nous utilisons le programme suivant :

```
class Composant(var id : Integer) is {

    var estEnPanne : Integer;

    def declencherPanne() is {
        estEnPanne := 1;
    }

    def reparer() is {
        if estEnPanne = 1 then {
            while composant.neFonctionnePas() = 1 do
                while composant.enStock() = 1 do
                    "changer le composant".println();

        }

        else {
            "n'est pas en panne".println();
            if numeroClient = 1 then {
                while appel.client() = 0 do {
                    if messagerie = 1 then {
                        "laisser un message".println();
                    }
                    else
                        "prendre rendez-vous".println();
                }

            }
            else
                "attendre".println();
        }
    }
}
{}
```

Au passage, ce programme permet de vérifier le bon fonctionnement des chaînes de caractères.

5.3 Opérateurs arithmétiques et concaténation de chaînes de caractères

Le langage BLOOD supporte les opérateurs arithmétiques suivants :

- Addition : +
- Soustraction : -
- Multiplication : *
- Division : /
- Concaténation de chaînes de caractères : &

Le programme utilisé est le suivant :

```
{
    x : Integer;
    y : Integer;
    z : Integer;
    str : String;
    is
    x := (3+2)/4;
    y := 3+2/4;
    z := 1 + 2 + 3 - 4 + 5 - 6;

    x := 8 * 2 - 4;
    y := (4 * (3 + 4))/(8 + 4);
    z := x / y;

    str := "Cette chaine" & " est " & "concatenee.";
    str := "Autre exemple avec une methode : " & c.getModelName();
}
```

Pour plus d'exemples d'AST nous vous invitons à lire le fichier `PCL1-intermediate.md`.

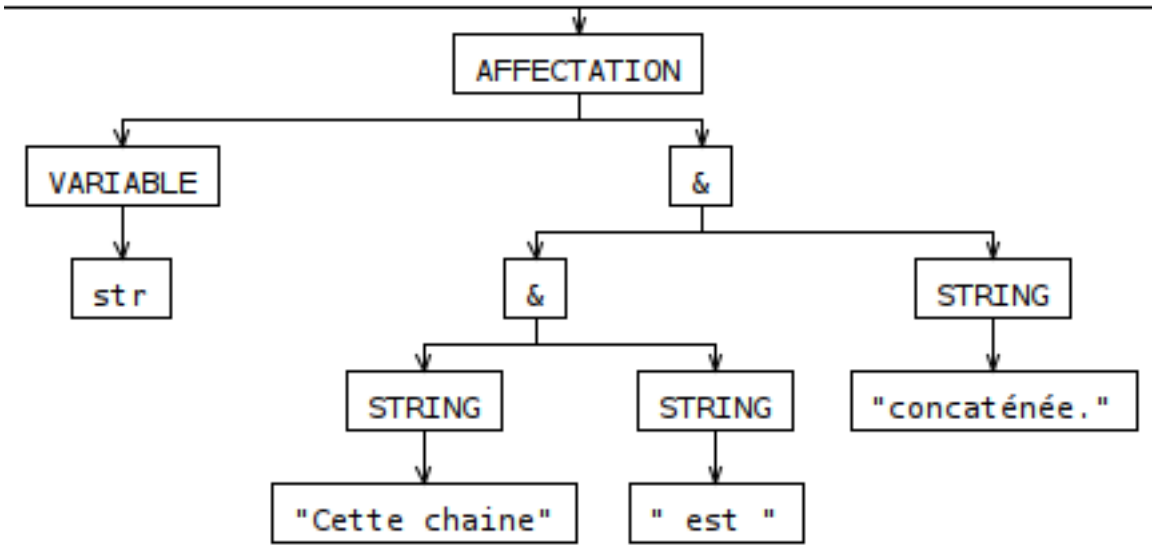


Figure 5.1 – Exemple de chaîne concaténée

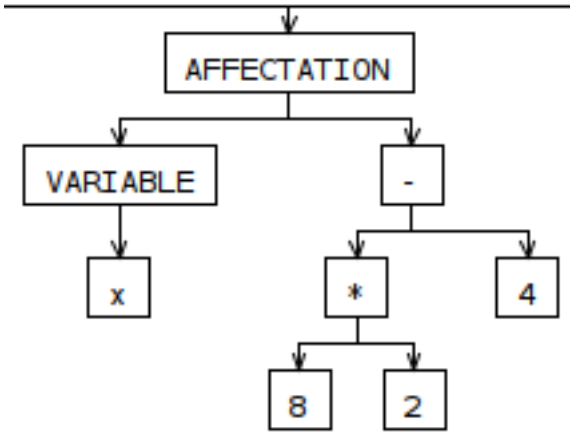


Figure 5.2 – Exemple d'utilisation d'opérateurs

En plus du fait que la structure doit bien être reconnue par la grammaire, ce test a aussi été important dans la validation de notre méthode pour l'affichage dans l'AST des opérations.
A noter que ce fichier permet aussi de tester le bloc principal et les messages. Cependant, ces derniers auront droit à des tests plus poussés dans la section suivante 5.4.

5.4 Messages

Les messages sont une fonctionnalité centrale d'un langage orienté objet. En effet, ce sont eux qui permettent, entre autres, l'appel des méthodes d'une classe. Le programme de test utilisé est le suivant :

```
{
    /*constructeur  vide*/
      moteur : Moteur := new Moteur();
    /*constructeur plusieurs arguments*/
      v : Vehicule := new Vehicule("Ford", 4, moteur);
      is
      v.setMarque(moteur.getMarque());
      v.setMarque(moteur.getMarque());
    /*appel sur un attribut*/
      v.moteur.getVitesseMax();
    /*appel de fonction multiple*/
      v.getMarque().getNationalite(param1, param2 , "vidange");
}
```

Ce programme permet encore une fois de vérifier le fonctionnement du bloc principal, ainsi que de l'instanciation. Aussi, il permet de tester les commentaires, balisés par "/*".

5.5 Bloc principal et instanciation

Les programmes écrits en BLOOD doivent impérativement contenir un bloc de code équivalent à une fonction `main`, contrairement aux classes qui elles sont facultatives. Le fichier suivant teste donc à la fois le bon fonctionnement de ce bloc et celui de l'instanciation de nouveaux objets.

```
{
    ordi : Ordinateur := new Ordinateur(0);
    cg : CarteGraphique := new CarteGraphique(0);
    proc : Processeur := new Processeur(1, 4, 4, "i5");
    ram1 : RAM := new RAM(2, 4, "DDR4");
    ram2 : RAM := new RAM(3, 4, "DDR4");
    is
    ordi.allumer();
    if proc.estEnPanne() = 1 then
```

```
10          "Le processeur est en panne !".println();
11      else
12          "Le processeur fonctionne".println();
13      while ram2.estEnPanne() = 1 do
14          "La RAM est en panne. Merci de reparer.".println();
15  }
```

6 Conclusion et Bilan

6.1 Conclusion

Ce premier rendu contient une grammaire ainsi que l'AST associé pour le langage BLOOD. La grammaire a été testée dans le chapitre 5. L'AST, lui, a été contrôlé de manière attentive dans le chapitre 4.
L'objectif de ce premier rendu, et c'est le but des contrôles et tests effectués, était avant tout de constituer une base solide pour la partie II de ce projet. En effet, ce rendu concerne uniquement la partie **front-end**, il agit comme point de départ de la partie consacrée au **coeur du compilateur** et au **back-end**. Nous avons ainsi gardé toujours à l'esprit, dans chacun de nos choix de conception, que ce que nous faisions devait pouvoir être utilisé facilement dans la partie II.
Nous avons eu parfois l'impression de résoudre des problèmes de manière facile puis d'être bloqués subitement. En réalité, la première impression était souvent trompeuse, et nous conduisait à faire trop rapidement des choix engageants pour la suite. Malgré tout, le rapport et le rendu qui vous parviennent tiennent compte de nos erreurs et des remarques qui nous ont été faites.
Nous retenons de cette première partie qu'il faut savoir avoir une vision large et sur le long terme de ce que représente le projet de compilation. Cette perception est nécessaire car elle permet d'avancer efficacement et non temporairement, en mêlant rigueur, pragmatisme et organisation.

6.2 Bilan des membres de l'équipe projet

6.2.1 Bilan individuel d'mask mask

Points positifs	J'apprécie beaucoup le sujet de ce projet car il nous permet de construire nous-même un outil dont on se sert régulièrement. Cela nous permet de comprendre réellement comment un compilateur fonctionne.
Expérience personnelle/ressenti	- Le fait que le travail sur **AntlrWorks** soit très visuel (grâce à l'option d'Anltr qui nous montre schématiquement chaque règle) m'a beaucoup aidé et m'a permis de mieux comprendre par quelles règles passait **Antlr** lorsqu'on lançait la grammaire sur un exemple. - La gestion du temps étant très différente entre les membres du groupe, cela a parfois généré des tensions, notamment si certains avaient besoin du travail d'une autre personne pour effectuer sa tâche. De plus, la communication n'était pas très efficace, ce qui a parfois compliqué le travail de chacun.
Axes d'amélioration	Une meilleure concertation avec les membres de l'équipe aurait permis une répartition du travail plus claire au sein du groupe, ce qui aurait permis d'éviter les quelques fois où deux personnes réalisaient la même tâche. Une meilleure communication serait aussi bénéfique.

6.2.2 Bilan individuel de moi moi

Points positifs	J'ai beaucoup aimé le sujet du projet. Il abordait un thème à la fois nouveau pour moi, et en même temps attendu. En effet, nous n'avions jamais eu à réfléchir à la façon dont les compilateurs que nous utilisions étaient faits. Par ailleurs, ce thème avait déjà été abordé en MI1-2, ainsi, il me tardait de passer à la phase pratique du thème.
Expérience personnelle/ressenti	— La difficulté du projet était liée à l'unicité du résultat attendu. — En effet, je me suis rapidement rendu compte que ce projet demandait un rendu extrêmement spécifique. Les choix d'implémentations sont déterminants et peuvent s'avérer catastrophiques s'ils sont mauvais. A ce titre, il nous est arrivé de devoir refaire plusieurs fois intégralement la grammaire pour des erreurs qui nous semblaient minimes mais s'avéraient non corrigibles facilement. — Il a été ainsi parfois difficile de se mettre d'accord sur la solution à implémenter. En effet, même si le rendu peut paraître très formaté, des interprétations différentes étaient possibles sur certains points. — La communication n'était pas fluide dans le groupe de messagerie instantanée mis en place. Souvent, des messages restaient non répondus pendant plusieurs jours, et m'ont conduit à prendre des initiatives qui se sont parfois révélées concerner le travail d'un autre membre du groupe.
Axes d'amélioration	Je souhaiterais qu'une convention de communication soit mise en place.

6.2.3 Bilan individuel de mask mask

Points positifs	Dans un sens, j'apprécie ce projet car il me permet de mettre en application des concepts vus en MI l'année précédente, et m'a donc permis de comprendre certains concepts avec lesquels j'avais du mal.
Expérience personnelle/ressenti	— Bien que le sujet me permette de mieux comprendre les concepts vus en première année et le fonctionnement interne d'un compilateur, j'ai beaucoup de mal à prendre du plaisir à travailler dessus, alors que dans les autres projets je n'avais aucun mal à faire ressortir des points positifs. J'imagine néanmoins que cela est formateur, dans le sens où tous les projets sur lesquels nous sommes amenés à travailler ne nous passionnerons pas forcément. — La communication n'était pas toujours évidente dans le groupe. — De manière générale, j'ai trouvé ce projet assez compliqué dans le sens où un mauvais choix de conception pouvait rendre certains points extrêmement difficiles à implémenter (nécessitant de ce fait une refonte presque complète de la grammaire par moment). De plus, **ANTLRworks** peut être particulièrement capricieux par moments, ce qui ne simplifie pas les choses.
Axes d'amélioration	— Plus de rigueur quant au respect des conventions de nommage et de commits (des fois c'est en anglais, des fois en français). — La communication.

6.2.4 Bilan individuel de mask mask

Points positifs	J'ai apprécié le thème du projet puisque jusqu'à celui-ci je ne m'étais jamais demandé comment fonctionne un compilateur. Ainsi, j'ai pu en apprendre beaucoup sur son fonctionnement et mettre en application des connaissances de première année dont j'avais du mal à comprendre l'utilité concrète.
Expérience personnelle/ressenti	— **ANTLRworks** est pour un outil à la fois très bon et mauvais, d'un coté il nous guide très bien grâce à la détection de grammaire non LL(1) et son debugger aide grandement à trouver les erreurs dans la grammaire comme dans l'AST. D'un autre coté, **ANTLRworks** est rempli d'erreurs qui nuisent à la conception ; la dernier en date m'empêche totalement de lancer le mode debugger (bug connu mais sans solution) ce qui m'a forcé à utiliser un deuxième ordinateur pour continuer le projet (heureusement que j'en avais un sous la main). — Travail et communication en groupe pas toujours evidents. — Le confinement et le travail à distance n'a de plus rien arrangé, même si le travail a pu être accompli, cela a rajouté un poids sur un travail pas forcément facile et que je n'appréciais pas particulièrement.
Axes d'amélioration	— Des réunions qui débordent du sujet, je pense qu'elles doivent être faites pour informer le reste du groupe du travail effectué. — Communication, cela rejoint un peu l'item précédent.

6.3 Travail réalisé

Étapes	mask	moi	mask	mask
Jalons				
0 Début du projet	1h	1h	2h	1h
1 Grammaire	16h	18h	8h	22h
2 AST	24h	20h	18h	10h
3 Mise en forme du dépôt pour la soutenance	3h	3h	10h	2h
4 Prise en compte des remarques suite à la soutenance	2h	1h	7h	2h
Travail de secrétariat				
Compte-Rendus de réunion	0h	4h	1h	0h
Rapport	6h	6h	5h	3h
Total	52h	53h	51h	40h

Bibliographie / Webographie

[1] IONOS. Compilateur et interpréteur : explications et différence, 2020. https ://www.ionos.fr/digitalguide/sites-internet/developpement-web/compilateur-vs-interpreteur/. xiv

[2] Jacques Ferber. Cours de compilation - 1ère partie -, 1997. http ://www.lirmm.fr/ ferber/Compilation/compil1.htm. xiv

Annexes : Compte-Rendus de réunion

A CR Réunion 12/10/2020

Présents :

- mask mask
- moi moi
- mask mask
- mask mask

Heure de début : 16h

Durée : 1h30

Ordre du jour :

- Prise de contact
- Choix du chef de projet
- Création du dépôt git
- Planification des premiers objectifs au regard du sujet : grammaire et AST

I Prise de contact, retour sur la perception du sujet (incompréhensions, but recherché), et retour sur le TP1

Les grandes lignes du sujet ont été dégrossies. Chacun était invité avant la réunion à lire attentivement le sujet et y noter ses incompréhensions et impressions. Voici le compte rendu des échanges :

- Le sujet ne fait pas mention des imports (traditionnels en Java par exemple), on ne s'attachera donc pas à les intégrer à notre grammaire.
- La casse des noms de fonctions et de classe doit respecter des règles précises :
  "Les noms de classes doivent débuter par une majuscule ; tous les autres identificateurs doivent débuter par une minuscule. Les mots-clefs sont en minuscule. La casse des caractères importe dans les comparaisons entre identificateurs."
- Le terme "d'envoi de messages" dans *"Les objets communiquent par "envois de messages". Un message est composé du nom d'une méthode avec ses arguments ; il est envoyé à l'objet destinataire qui exécute le corps de la méthode et peut renvoyer un résultat à l'appelant."* paraissait peu clair.
  Nous en avons conclu que cela signifiait simplement que l'appel à des méthodes et la récupération du résultat de cet appel était possible.
- *"une expression suivie d'un ; a le statut d'une instruction : on ignore le résultat fourni par l'expression."* Ce fonctionnement semble s'apparenter au fonctionnement de `Matlab`...

— *"La syntaxe d'un paramètre a la forme suivante :* `[var] nom : nomClasse`
S'il est précédé du mot clef `var`*, un paramètre définit implicitement un attribut d'instance de la classe qui sera automatiquement initialisé par la valeur de l'argument fourni à l'appel du constructeur de la classe".*
Après réflexion, cela semble indiquer que tout paramètre précédé de `var` est une variable de l'instance de classe entière.

II Choix du chef de projet

Le chef de projet est mask mask.

III Création du dépôt git

Le dépôt git a été créé par mask mask, il l'a réalisé en fonction des spécifications du sujet.

IV Plannification des premiers objectifs (Grammaire, AST, TDS)

Rappel du sujet *Vous définirez la grammaire LL(1) du langage et la soumettrez à* ***Antlr*** *afin qu'il génère l'analyseur syntaxique descendant. Bien sûr, l'étape d'analyse lexicale est réalisée parallèlement à l'analyse syntaxique. Vous aurez testé votre grammaire sur des exemples variés de programmes écrits en langage BLOOD, avec et sans erreurs lexicales et syntaxiques. Vous réfléchirez ensuite à la construction de l'arbre abstrait que vous mettrez en oeuvre. Comme mentionné précédemment, une démonstration de l'AST (construit à partir d'une grammaire LL(1)) sera faite lors de la séance TP4. Vous commencerez alors à réfléchir à la mise en oeuvre de la TDS.*

Dates des livrables *Semaine 49 (30 novembre au 4 décembre), lors de la séance TP4 au cours de laquelle vous nous présenterez la grammaire et l'AST. Vous rendrez pour le 15 janvier un premier dossier (cf. rubrique Les dossiers pour en connaître le contenu). Cette évaluation et le dossier comptent pour la validation du module PCL1.*

Au vu des demandes du sujet, il a été décidé de commencer par l'écriture de la grammaire. Pour accélérer cette conception, le travail sera partagé entre tous les membres de l'équipe. Pour commencer, nous nous attacherons à reconnaître les principaux constituants d'une grammaire : `class`, méthodes, mots clefs... Le travail à effectuer par chacun pour la prochaine réunion est détaillé dans **Pour la prochaine fois**.

L'ordre du jour étant épuisé, la réunion s'est terminée à 17h30

Prochaine réunion : le jeudi 22/10 12H

Pour la prochaine fois :

— Noter chacun les éléments que la grammaire devra encore reconnaître qui n'ont pas été cités ci-après :

mask mask	moi moi	mask mask	mask mask
Nommage (reconnaissance des noms de classes, autres identificateurs et mots clefs.)	Reconnaissance de classes	Reconnaissance de méthodes	Attributs, Expressions et Instructions (expression, bloc , `return`, cible :=expression et `if`)

Chacun est ainsi responsable de la construction d'un ensemble de règles et de leur implémentation dans un fichier `.g`.

Eléments post réunion En TD de TRAD, une grammaire pouvant correspondre à la grammaire que nous sommes en train de développer a été donnée, il s'agira donc de s'en inspirer pour réaliser les éléments à faire pour la prochaine réunion.

B CR Réunion 22/10/2020

Présents :

- mask mask
- moi moi
- mask mask
- mask mask

Heure de début : 12h

Durée : 1h30

Ordre du jour :

- Retour sur le jalon effectué depuis la dernière réunion : construction de la grammaire, début des tests
- Les difficultés rencontrées :
  - les types prédéfinis `Integer` et `String`
  - la détection du `return`
  - les sélections
  - détection de plusieurs classes
- Problème d'ambiguïté dans la grammaire

I Retour sur le jalon effectué depuis la dernière réunion : construction de la grammaire, début des tests

Construction de la grammaire Chaque membre s'est attelé à rajouter dans la grammaire les règles qu'il devait développer.

Début des tests La phase de tests a commencé. A ce titre un fichier a été créé contenant un exemple de fichier écrit en BLOOD. Le fichier est situé dans `\test` et s'appelle `test.txt`. Ce fichier, à améliorer en continu, a déjà permis de mettre en lumière quelques ratés dans la grammaire.

II Les difficultés rencontrées

Les types prédéfinis Integer et String Nous avons rencontré un problème avec la première version de la grammaire. Le sujet nous informe que les types `Integer` et `String` sont des types implémentés de base dans tout programme en BLOOD. La

difficulté est que la présence de deux règles pouvant mener à la reconnaissance d'`Integer` et `String` génère des erreurs dans Antlr :

$$NameRule \rightarrow' Integer'$$
$$NameRule \rightarrow' String'$$
$$ClassId \rightarrow ('A'..'Z')('a'..'z'|'A'..'Z'|'0'..'9')*$$

Nous avons ainsi choisi de supprimer les règles

$$NameRule \rightarrow' Integer'$$
$$NameRule \rightarrow' String'$$

La détection du mot clé return Le sujet précise que certains types de méthodes doivent impérativement comporter un `return`. Nous nous sommes alors demandé s'il fallait que notre grammaire comporte un système de vérification de la présence du `return`.
Réponse des responsables du projet : la grammaire doit rester simple et ce genre de vérifications alourdit trop. La grammaire doit seulement permettre de reconnaître le texte, pas d'en vérifier la cohérence. Il en est de même avec les constructeurs, il n'est pas nécessaire que la grammaire comporte une vérification pour attester qu'ils sont présents, mais la grammaire doit les reconnaître.

Les sélections Le sujet précise qu'un programme peut comporter une entrée du type :
`index := Point.cptPoint.incr();`
Ce cas de figure n'était pas prévu dans la grammaire : il s'agit de modifier la règle des sélections pour qu'une sélection puisse commencer avec un nom de classe.

Détection de plusieurs classes Nous nous sommes également interrogés sur le fait que le sujet comporte un exemple avec plusieurs classes. Un fichier BLOOD peut-il contenir plusieurs classes ? Nous demanderons aux professeurs concernés des précisions sur ce point.

III Problème d'ambiguïté dans la grammaire

Au cours d'un test réalisé pendant la réunion nous nous sommes aperçus que **AntlrWorks** permet l'affichage des ambiguïtés (un même programme peut être

reconnu par plusieurs chemins) de la grammaire. Notre grammaire contenant des ambiguïtés, un des prochains jalons sera de les résoudre.

Prochaine réunion : le 31/10

Pour la prochaine fois :

mask mask	moi moi	mask mask	mask mask
Continuer le fichier de tests	Solution des ambiguïtés de la grammaire	Ajout à la grammaire de `while`	Solution des ambiguïtés de la grammaire

Eléments post réunion Les professeurs responsables du projet ont donné un exemple de fichier BLOOD. Celui-ci montre que plusieurs classes peuvent être présentes au sein d'un même fichier ce qui n'est pas possible avec notre grammaire pour le moment. Il convient donc de la corriger.

Pour la prochaine réunion [correctif] : Jeudi 22/10 22H

Pour la prochaine fois :

mask mask	moi moi	mask mask	mask mask
-Ajouter le fichier de tests fourni sur l'ENT -Modification de la grammaire pour détecter plusieurs classes et le bloc d'instruction	Détection des `String`	Génération de l'AST et simplification de la grammaire	Vérifier que la nouvelle grammaire est complète

C CR Réunion 31/10/2020

Date : Samedi 31 octobre 2020

Présents :

- mask mask
- moi moi
- mask mask
- mask mask

Heure de début : 15h

Durée : 1h30

Ordre du jour :

- Retour sur le jalon effectué depuis la dernière réunion :
- Éléments à compléter dans la grammaire
  La détection :
  - des String
  - du bloc comportant `'is'`
  - du mot clé `'return'`
  - des éléments négatifs (ex : `if(-9<-3)` n'est pas encore reconnu)
  - des commentaires
  - de `'this'` et `super`
- Explication sur la façon de faire un AST avec **Antlr** et exemples

I Retour sur le jalon effectué depuis la dernière réunion

Chacun expose à tour de rôle son travail depuis la dernière réunion :

- mask mask : A effectué des recherches sur la façon de générer l'AST de la grammaire et ajouté la détection de `while` à la grammaire.
- moi moi : A commencé la détection des `String` et a participé à la refonte de la grammaire (pour obtenir la grammaire LL(1)).
- mask mask : A ajouté le fichier `.blood` présent sur l'ENT en prenant soin d'enlever les commentaires et a modifié la grammaire pour la détection d'un fichier de la forme : class* bloc (modification du fichier de test avec maintenant plusieurs classes et un unique bloc d'instructions).
- mask mask : A vérifié que la grammaire était complète et LL(1).

II La détection des String

moi moirevient sur les difficultés qu'il a rencontrées dans la détection des `String`.
Les `String` sont assez difficiles à détecter car il ne s'agit pas simplement de chaînes du type : `aux : String := "Blanc"`.
On peut par exemple avoir :
`aux : String := "Blanc d' oeuf: ajouter-les! & battez_les vivEment`
De plus, il faut, le sujet le demande d'ajouter la possibilité de concaténer des chaînes de caractères, par exemple :
`"Blanc d' oeuf: ajouter-les!" & suite_recette()` Avec `suite_recette()` une fonction qui retourne un `String`
Ainsi moi moin'a pas encore pu finir la détection des `String`.

III La détection du bloc comportant 'is'

Le bloc avec le mot `is` au milieu (liste non vide de déclaration de variables + 'is' + liste non vide d'instructions) n'est pour le moment pas encore détecté, il convient de procéder à l'élaboration de la règle.

IV La détection du mot 'return'

mask masks'est chargé de la détection de `return`, d'après le sujet `return` peut se trouver uniquement sous cette forme :

```
1  def manger(var quantite: LongInteger) : Integer is {
2      result := new Nourriture();
3      if (-manger || burger) then {
4          if(pasta && manger) then {} else {return;}
5      }
6      else {b := -b;}
7          return;
8      }
9  }
10
```

V La détection des éléments négatifs

Nous nous sommes rendu compte d'un problème : `if(-9<-3)` et `if(9<-3)` ne sont pas encore reconnus
par contre : `if(-9<3)` l'est.
Il convient donc de donner la possibilité aux seconds (et plus) membres d'une expression d'être négatifs ; on pourrait par exemple avoir
`b1 : Integer := - p.getx() - this.x - 38;`

VI La détection des commentaires

Les commentaires avaient été retirés manuellement du fichier `.blood`, toutefois après vérification dans le cours de TRAD1, le rôle de l'analyseur syntaxique est bien, entre autres, d'enlever automatiquement les commentaires. Il faudra donc les ajouter au fichier de tests et créer une règle de reconnaissance des commentaires.

VII La détection de 'this' et 'super'

Les mots `this` et `super` ne sont pour le moment pas encore détectés. Il convient de procéder à l'élaboration des règles pour permettre leur reconnaissance. Le sujet précise bien que *"les identificateurs ont le même sens qu'en Java*, on peut donc avoir :

```
1  class manger() is {
2      var static maConstante : Integer := 0;
3      def Manger(var quantite: LongInteger) : Integer is {
4          super(quantite);
5          this.result := new Nourriture();
6          if (-manger || burger) then {
7              if(pasta && manger) then {} else {return;}
8          }
9          else {b := -b;}
10         this.maConstante := new Nourriture(this);
11         return;
12     }
13 }
14
```

VIII Explication sur la façon de faire un AST avec Antlr et exemples

mask maskprésente le résultat de ses recherches. Construire un AST est assez complexe.
Il s'agit de reprendre chaque règle de grammaire et d'indiquer à **Antlr** quel est le père d'un noeud et quels sont les fils, cela en adoptant une syntaxe particulière.
Ainsi, pour avoir un arbre, il faut déjà avoir une grammaire complète. Malgré tout mask maska commencé la réalisation de l'arbre à partir de la grammaire existante.

Prochaine réunion : le 25/11

Pour la prochaine fois :

mask mask	moi moi	mask mask	mask mask
La détection des éléments négatifs -Ajouter au fichier de test des exemples pour vérifier la détection des éléments négatifs	-Finir la détection des `String` -Vérifier le fichier `.blood` (des erreurs sont apparues pendant la réunion) -La détection des indicateurs `this` et `super`	Simplifier la grammaire et commencer l'AST	-La détection du bloc avec 'is' -Détection des commentaires -Ajouter les commentaires au `.blood`

D CR Réunion 25/11/2020

Présents :

- mask mask
- moi moi
- mask mask
- mask mask

Heure de début : 13h30

Durée : 1h30

Ordre du jour :

- Retour sur le jalon effectué depuis la dernière réunion :
  - Le fichier de tests (`.blood` et le nôtre)
  - La détection des `String`
  - La détection des indicateurs `this` et `super`
  - Détection des commentaires
  - Rationalisation des opérateurs
  - AST
- Perfectionnement de l'AST
  - Retrait du noeud message quand non nécessaire
  - Retrait des " " autour du terminal de `String`
- Mise en forme du dépôt pour le rendu (fichiers tests et `.MD`)

Retour sur le jalon effectué depuis la dernière réunion

Le fichier de tests (.blood et le nôtre) Le fichier `.blood` et le lot de tests utilisé comportaient plusieurs erreurs dues à l'utilisation d'un OCR -pour la traduction en fichier exploitable- sur le fichier .pdf fourni par les encadrants.
Les erreurs étaient de plusieurs types et la plupart du temps facilement identifiables : oublis de ')' '"' '.'
Par contre, une erreur très difficilement décelable s'était glissée dans les fichiers.

```
1 class Test() is {
2     def Test() is {}
3     def static test(p: Point, p2: PointColore, p3: PointNoir) is {
4         c: String; c2: String; c3: String;
5         true: Integer := 1;
6         false: Integer := 0;
7         is
8         p.print(true);
9         p2.print(true);
```

```
10          if p2.colore()  <> 0 then c := "colore"; else c := "gris";
11      }
12
13      def static test2(p: PointColore) is {
14          p.couleur().name(0).print();
15      }
16  }
17
```

dans cet exemple, les lignes `p.print(true);` et `p2.print(true);` n'étaient jamais reconnues par **Antlr**. En fait, le problème venait du fait que le caractère `p` semblait être un caractère de l'alphabet usuel, sauf que ce n'était pas le cas !

La détection des String La règle de détection des String était incorrecte.

```
1  string: (ID | CLASSID|symboles )* -> ^(STRING ID* CLASSID*)
2
```

cette implantation conduisait à des erreurs dans l'AST. Celui-ci décomposait une `String` en `ID`, `CLASSID` et `symboles`. De plus, la règle précédente incluait seulement "approximativement" la reconnaissance des symboles.
Maintenant, la règle est un terminal et inclus par défaut toutes les possibilités de `String`.

```
1              STR : '"'((~('"')  )*)'"';
2
```

La détection des indicateurs this et super `this` et `super` ont été rajoutés à la grammaire. Par contre, `result` n'est toujours pas dans la grammaire.
Il faudra réfléchir à un réagencement possible de certaines règles pour créer une règle `identificateur` comme le demande le sujet.

Détection des commentaires Les commentaires sont maintenant reconnus par la grammaire, le fichier de tests comporte lui aussi des commentaires pour vérifier le bon fonctionnement de la règle.

```
1  COMMENT:    '/*' ( options {greedy=false;} : . )* '*/' {$channel=HIDDEN
       ;};
2
```

Rationalisation des opérateurs Suite à la réunion avec MME COLLIN, nous avons identifié un problème avec nos opérateurs.

```
1                p2.move(p1.getx()-2*5-3, p1.gety(), true)
2
```

n'était pas détecté

```
1                p2.move(p1.getx() - 2 * 5 - 3, p1.gety(), true)
2
```

était reconnu. Cela était en réalité le reflet d'une concurrence entre les opérateurs unaires et binaires '+' '-'.
MME COLLIN nous a alors conseillé de reprendre intégralement la construction des opérateurs. Ceux-ci ne respectaient pas les priorités opératoires et étaient en surnombre. Cela a été fait par mask mask.

AST mask masks'est penchée sur la finalisation de l'AST. Celui-ci est complet mais comporte toutefois certains détails à améliorer.

I Perfectionnement de l'AST

Message Le noeud `Message` apparaît de manière intempestive lorsqu'aucune fonction n'est appliquée sur un objet.

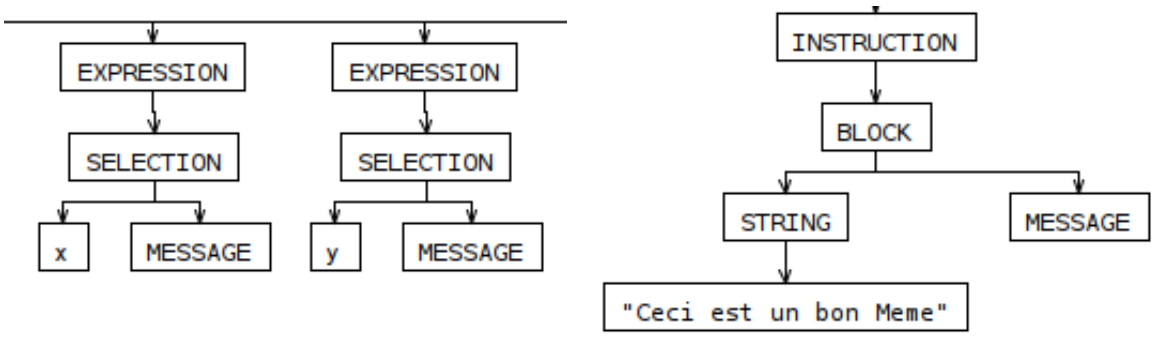


FIGURE D.1 – Mobilier(x,y) et "Ceci est un bon Meme"

String Apparition des " " dans le noeud d'un String

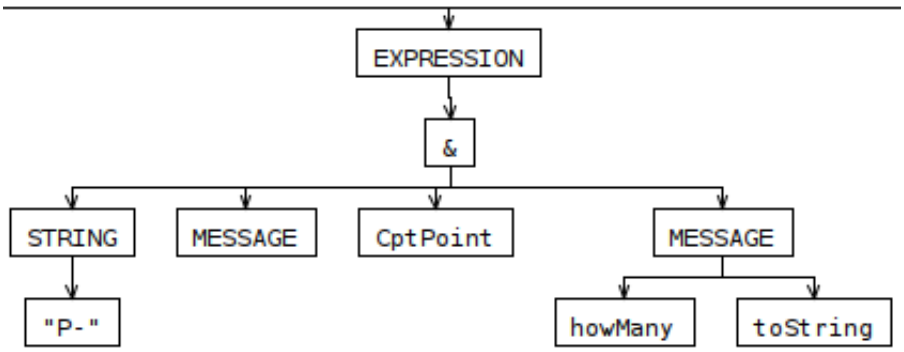


FIGURE D.2 – "P-" & CptPoint.howMany().toString())

II Mise en forme du dépôt pour le rendu (fichiers tests et .MD)

Nous avons reçu un mél de MME COLLIN demandant une mise en forme particulière du dépôt avec un `.MD` récapitulant des exemples reconnus. Le mél précisait aussi d'incorporer un jeu de tests pour la soutenance.

Chacun expose à tour de rôle son travail depuis la dernière réunion :

Prochaine réunion : Après la soutenance, à la rentrée.

Pour la prochaine fois :

mask mask	moi moi	mask mask	mask mask
-`.MD`	Rapport	Rapport	Correction des erreurs de l'AST (noeud Message intempestif et String)
-Jeu de tests pour la soutenance			

Eléments post réunion

E CR Réunion 06/01/2021

Présents :
- mask mask
- moi moi
- mask mask
- mask mask

Heure de début : 14H

Durée : 1h30

Ordre du jour :
- Prise en compte des remarques suite à la soutenance
  - Étiquette MESSAGE dédoublée
  - Remonter étiquette MESSAGE
  - Changement de l'étiquette ' :=' pour VALUE pour la définition d'une variable
  - Vérification de l'étiquette ' :=' dans le cas d'une affectation
  - Création de l'étiquette NEW
  - Suppression this implicite pour les fonctions
  - Ajout des étiquettes IF (CONDITION), THEN, ELSE
- Rapport : Affectation des éléments manquants et discussion sur leur contenu
  - Chapitre : Grammaire du langage
  - Chapitre : Structure de l'arbre abstrait
  - Chapitre : Jeux d'essais
  - Relecture et complétion

I Prise en compte des remarques suite à la soutenance

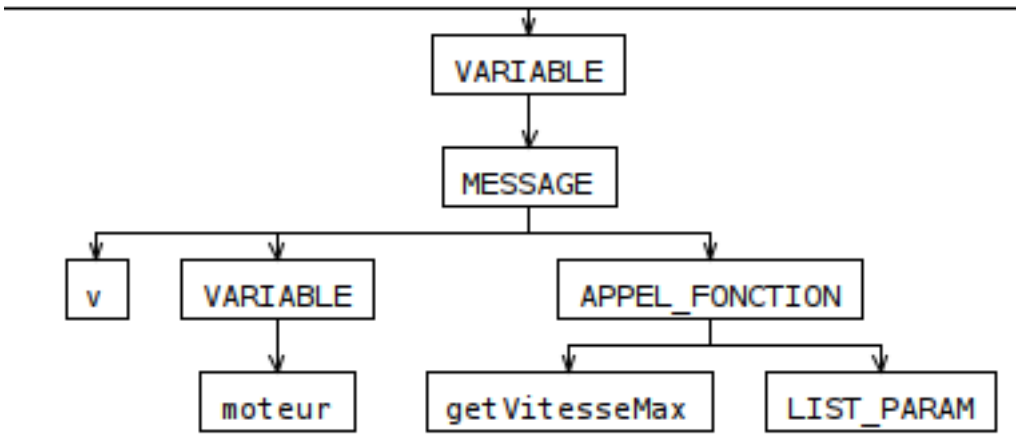
Chacun a pu effectuer les tâches prévues dans la matrice RACI 1.2.

Points encore à régler Après vérification mutuelle des différents points effectués, il reste encore quelques erreurs à régler dans l'AST.

Pour :

```
1 v.moteur.getVitesseMax();
2
```

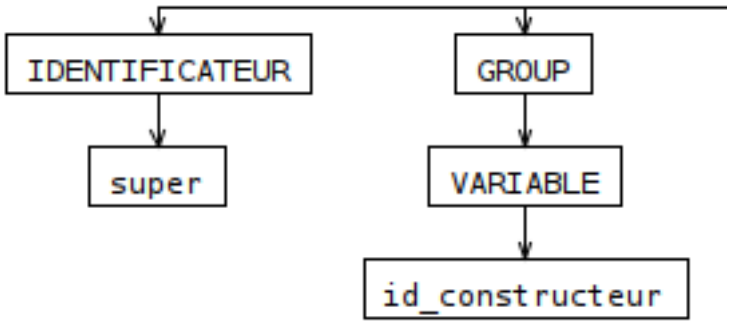
Il conviendrait davantage de mettre des niveaux d'AST distincts correspondants aux différents niveaux d'imbrication des sélections/appels/messages.



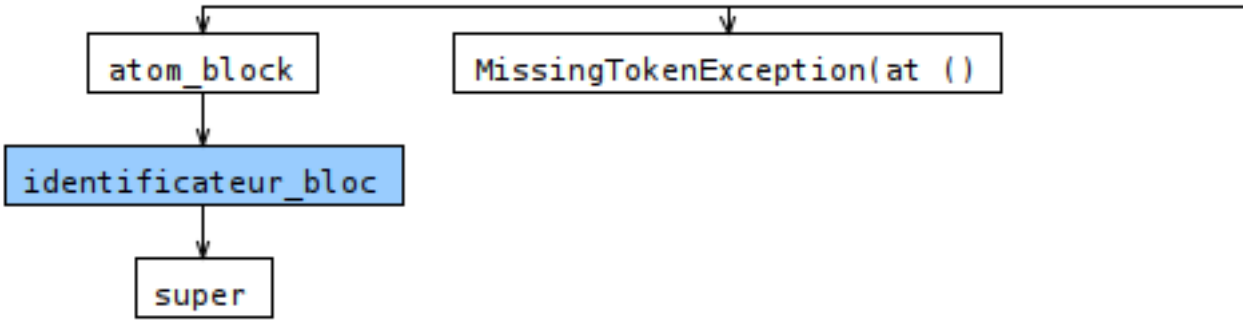
2 problèmes à résoudre pour :

```
1 super(id_constructeur);
2
```

— Il faut relier `super` à son argument.



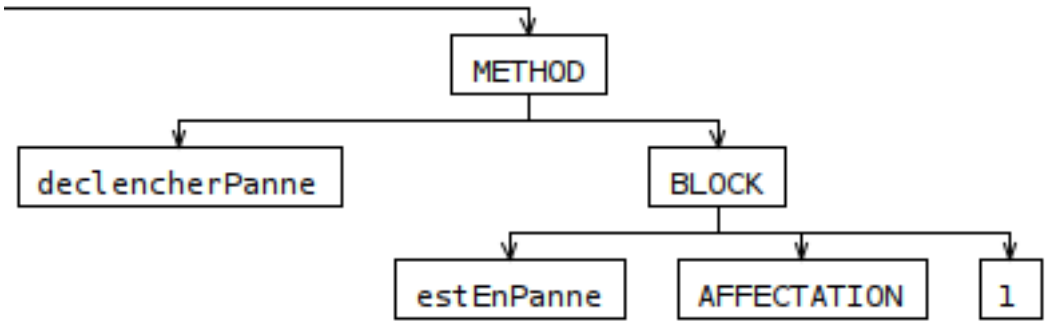
— Il faut résoudre le problème de missing token exception.



Pour :

```
1 estEnPanne := 1;
2
```

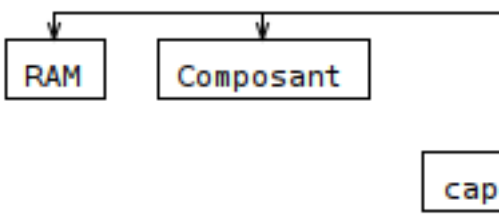
Il faut relier AFFECTATION à ses 2 fils.



Pour :

```
1             class RAM() extends Composant is
2
```

Il faut créer un noeuds EXTENDS pour pouvoir s'y retrouver facilement parmi les

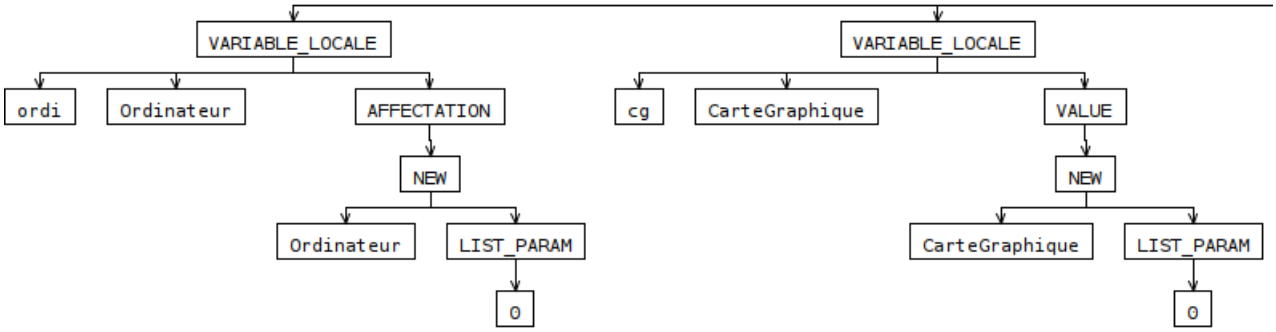


tokens et simplifier notre travail par la suite.

Pour :

```
1 ordi : Ordinateur := new Ordinateur(0);
2 cg : CarteGraphique := new CarteGraphique(0);
3
```

Les deux lignes sont équivalentes, il n'est donc pas normal que les étiquettes ne soient pas identiques.



Ainsi, nous faisons un point sur les étiquettes VALUE et AFFECTATION.
Nous décidons que :

- ATTRIBUT : est créée
  Son rôle est d'apparaître dans les lignes ci-après pour signaler une déclaration de variable en dehors du corps d'une méthode.

```
1 class Ordinateur(var id : Integer) is {
2     var carteGraphique  : CarteGraphique;
3     var processeur      : Processeur;
4 }
5
```

- VALUE
  Correspond à une affectation dans 2 cas précis :
  - Dans le cas d'une affectation d'une variable en dehors du corps d'une méthode (cas de l'étiquette ATTRIBUT).

```
1 class Ordinateur(var id : Integer) is {
2     var carteGraphique  : CarteGraphique := Seb;
3 }
4
```

  - Dans le cas d'une affectation conjointe à une définition dans le corps d'une méthode.

```
1 def allumage() is {
```

```
2       ordi : Ordinateur := new Ordinateur(0);
3
4       is
5
6       ordi.allumer();
7   }
8
```

— AFFECTATION
Correspond aux affectations sans définition immédiate (donc avec définition préalable).

II Rapport : Affectation des éléments manquants et discussion sur leur contenu

Chapitre : Grammaire du langage Nous convenons ensemble que ce chapitre devra notamment traiter de :

— Choix de la grammaire.

— Choix de conception.

— Du caractère LL(1) de notre grammaire

Chapitre : Structure de l'arbre abstrait Nous convenons ensemble que ce chapitre devra notamment traiter de :

— Choix effectués.

— Structure de l'AST.
Avec notamment une explication des relations père/fils entre les étiquettes.

Chapitre : Jeux d'essais Ce chapitre reprendra en substance, et après correction, les éléments présentés à la soutenance.

Relecture et complétion Une relecture individuelle sera effectuée pour limiter les risques d'erreurs.
Il reste à compléter individuellement :

— Les bilans personnels.

— Le nombre d'heures effectuées sur chaque tâche.

Prochaine réunion : Avant le début des cours de TRAD2 : le 25/01. Une réunion pourrait par exemple être organisée le 20 ou 21/01.

Pour la prochaine fois :

mask mask	moi moi	mask mask	mask mask
-Correction des erreurs concernant `super` (Parse Tree et AST)	-Ajout de l'étiquette EXTENDS à l'AST		
-Ajout de l'étiquette DECLARATION, correction des utilisations de VALUE et AFFECTATION			
-Correction du problème de niveaux en cas de messages multiples à la suite			
-Chapitre : Jeux d'essais	-Finition du Gantt	-Chapitre : Structure de l'arbre astrait	Chapitre : Grammaire du langage
	-Rédaction de la conclusion	-Suppression des étiquettes inutiles dans l'AST	