



TELECOM Nancy

PCL2

Projet Compilation - Langage Blood

Membres projet :
Marie-Astrid Chanteloup
Lucille Delaporte
Bleunwenn Graindorge
Niels Tilch

Responsables de module :
Suzanne Collin
Gerald Oster
Sebastien Da Silva



Table des matières

1	Introduction	5
1.1	Fiche Projet	5
1.1.1	Contexte du projet	5
1.1.2	Objectif et intérêt du projet	5
1.1.3	Rédacteurs, commanditaires	5
1.1.4	Présentation générale du projet	5
1.1.5	Livrables	5
1.1.6	Date limite	6
1.2	Organisation du document	6
1.3	Précisions légales	6
1.4	Clause de non-plagiat	6
2	Gestion de projet	11
2.1	Équipe de projet	11
2.2	Analyse du projet	11
2.2.1	Définition des objectifs	11
2.2.2	Analyse des risques : Matrice SWOT	11
2.3	Organisation du projet	12
2.4	Outils de travail	14
2.4.1	Réunion	14
2.4.2	Partage du travail	14
2.4.3	Rédaction du rapport	14
2.5	Comptes-rendu des réunions	15
2.5.1	Réunion du 15 janvier 2021	15
2.5.2	Réunion du 27 janvier 2021	15
2.5.3	Réunion du 3 février 2021	16
2.5.4	Réunion du 9 février 2021	17
2.5.5	Réunion du 17 février 2021	18
2.5.6	Réunion du 3 mars 2021	19
2.5.7	Réunion du 17 mars 2021	19
2.5.8	Réunion du 24 mars 2021	20
2.5.9	Réunion du 7 avril 2021	21
2.5.10	Réunion du 14 avril 2021	22
3	Table des Symboles	23
3.1	Organisation de la TDS	23
3.1.1	Définition de la TDS	23
3.1.2	Éléments présents dans une TDS	24
3.1.3	Traitement et affichage de ces éléments	24
4	Contrôle Sémantique	27
4.1	Analyse sémantique des classes	27
4.1.1	Déclaration	27
4.1.2	Gestion du <i>this</i>	27
4.2	Analyse sémantique des paramètres, attributs et variables	27
4.2.1	Déclarations	27
4.2.2	Instanciation	27
4.2.3	Affectations	27
4.3	Analyse sémantique des méthodes	28
4.3.1	Constructeur	28
4.3.2	Définition de méthode	28
4.4	Analyse sémantique des blocs	28
4.4.1	Gestion du <i>is</i>	28
4.4.2	Gestion du <i>result</i>	28
4.5	Analyse sémantique des opérations	28

- 5 Génération de code 29
 - 5.1 Principe de la génération de code 29
 - 5.2 Conventions et choix dans notre gestion du générateur 29
 - 5.2.1 Conventions sur les pointeurs, exceptions et adresses 29
 - 5.2.2 Convention sur le chaînage 30
 - 5.2.3 Préparation du *Main* 30
 - 5.3 Implémentation du niveau 1 : programme principal 30
 - 5.3.1 Déclarations de variables 30
 - 5.3.2 Récupération de variables 30
 - 5.3.3 Affectations simples et affectations d’expressions arithmétiques 30
 - 5.3.4 Expressions arithmétiques 30
 - 5.3.5 Structures de contrôles 31
 - 5.4 Implémentation du niveau 2 : Gestion des méthodes 32
 - 5.4.1 Méthodes *print*, *println* et *toString* 32
 - 5.4.2 Déclaration des méthodes 32
 - 5.4.3 Appels de méthode : gestion des messages 33
 - 5.5 Implémentation du niveau 3 : gestion des objets 33
 - 5.5.1 Allocation dans le tas 33
 - 5.5.2 Les constructeurs 33
 - 5.5.3 Les descripteurs de classe 33
 - 5.6 Implémentation du niveau 4 33
 - 5.6.1 Héritage 33
 - 5.6.2 Polymorphisme 34
- 6 Bilan du projet 35
 - 6.1 Rapport d’étonnement 2 - Février 2021 35
 - 6.1.1 Marie-Astrid CHANTELOUP 35
 - 6.1.2 Lucille DELAPORTE 35
 - 6.1.3 Bleunwenn GRAINDORGE 35
 - 6.1.4 Niels TILCH 35
 - 6.2 Bilan de la deuxième partie du projet 36
 - 6.3 Bilan du projet par membre 37
 - 6.3.1 Marie-Astrid CHANTELOUP 37
 - 6.3.2 Lucille DELAPORTE 37
 - 6.3.3 Bleunwenn GRAINDORGE 37
 - 6.3.4 Niels TILCH 37
 - 6.4 Travail réalisé 38

Chapitre 1

Introduction

1.1 Fiche Projet

1.1.1 Contexte du projet

Ce projet a été réalisé dans le cadre des modules PCL1 et PCL2 de la deuxième année du cycle ingénieur sous statut étudiant de TELECOM Nancy.

L'objectif est de concevoir un compilateur pour le langage Blood

Ce projet est scindé en 2 parties. La première concerne la construction de la grammaire et de l'AST (Abstract Syntax Tree, ou arbre syntaxique abstrait en français) associés au langage. La seconde partie concerne la construction de la table des symboles, la mise en place de tests sémantique et le développement du compilateur final du langage Blood.

1.1.2 Objectif et intérêt du projet

Le projet de Compilation permet de mettre en pratique et d'approfondir les compétences acquises en cours du module de Traduction des Langages et de l'ensemble des modules de 1A. L'objectif de ce projet est de réaliser un compilateur produisant du code assembleur pour un langage orienté objet.

1.1.3 Rédacteurs, commanditaires

Ce document a été rédigé par CHANTELOUP Marie-Astrid, DELAPORTE Lucille, GRAINDORGE Bleunwenn et TILCH Niels, les quatre membres de l'équipe projet. Le projet quant à lui a été commandité par les membres de l'équipe pédagogique du projet de Compilation qui est au nombre de 2 : COLLIN Suzanne, OSTER Gerald.

1.1.4 Présentation générale du projet

Le sujet du projet Compilation s'inspire d'un travail proposé aux étudiants de l'Université Paris-Saclay. L'objectif est d'écrire le compilateur du langage Blood. Le travail sera décomposé en deux grandes parties. Pour la première partie, appelée PCL1, il sera nécessaire d'utiliser l'outil ANTLR afin de réaliser la définition complète d'une grammaire et un arbre abstrait. Il faudra s'assurer que la grammaire est bien LL(1) et la tester sur des exemples de programmes.

Lors de la deuxième partie du projet, dite PCL2, il faudra construire la table des symboles, réaliser les contrôles sémantiques et générer le code assembleur correspondant. Le code généré devra être en langage d'assemblage *microPIUP/ASM*.

1.1.5 Livrables

Ce projet étant évalué, l'équipe projet devra rendre un certain nombre de livrables, voici la liste exhaustive de ceux de la première partie du module :

- La grammaire du langage
- La structure de l'arbre abstrait
- Des jeux d'essais permettant d'illustrer le fonctionnement et les limites de la grammaire et de l'arbre abstrait
- Les outils de gestion de projet

Le tout devra être rendu dans un rapport rédigé par l'ensemble du groupe.

Les livrables du module de PCL2 sont les suivants :

- La structure de la table des symboles
- Les erreurs sémantiques reconnues et traitées par le compilateur
- Des schémas de traduction vers le langage assembleur
- Des jeux d'essais supplémentaires
- Les outils de gestion de projet

1.1.6 Date limite

Le projet étant divisé en deux parties, il y aura deux rapports à rendre. Un premier à la fin du module PCL1 pour le 15 janvier 2021. La fin du module PCL2, correspondant à la fin du projet, est fixée au 16 avril 2021.

1.2 Organisation du document

Ce document rapporte la deuxième partie du projet.

Dans le chapitre 2, nous présentons les éléments ainsi que les outils de gestion de projet que nous avons utilisés.

Dans le chapitre 3, nous présentons la conception de la table des symboles à partir de l'AST réalisé en PCL1.

Dans le chapitre 4, nous présentons la mise en place de nos tests sémantiques à l'aide de l'AST et de la TDS.

Dans le chapitre 5, nous présentons le fonctionnement de la génération de code.

Dans le chapitre 6, nous avons consigné les rapports d'étonnement de février 2021 ainsi que le bilan de la seconde partie du projet, d'un point de vue personnel et global.

1.3 Précisions légales

Ce projet n'est pas destiné à un usage commercial, ainsi, les images présentes, notamment les images de test, ne sont pas destinées à la publication et ne sont pas toutes libre de droit.

Cependant, le caractère strictement scolaire de ce projet nous autorise à les inclure en accord avec :

- Code civil : articles 7 à 15, article 9 : respect de la vie privée
- Code pénal : articles 226-1 à 226-7 : atteinte à la vie privée
- Code de procédure civile : articles 484 à 492-1 : procédure de référé
- Loi n78-17 du 6 janvier 1978 : Informatique et libertés, Article 38

1.4 Clause de non-plagiat

Les déclarations sur l'honneur de non-plagiat des membres de l'équipe projet sont présentes dans les quatre pages suivantes.

Déclaration sur l'honneur de non-plagiat

Je, soussigné(e),

Nom, prénom : Chanteloup Marie-Astrid

Élève ingénieur(e) régulièrement inscrit en 2^{me} année à TELECOM Nancy

Numéro de carte étudiante : 31916886

Année universitaire : 2020/2021

Auteur, en collaboration avec Delaporte Lucille, Graindorge Bleunwenn et Tilch Niels du rapport :

Projet Compilation - Langage Blood

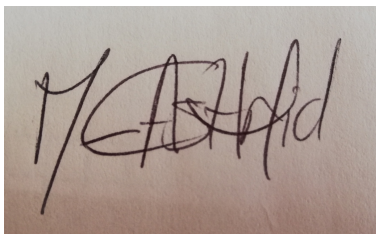
Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

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 Nancy, le
Signature : 17/04/2021

A photograph of a handwritten signature in dark ink on a light-colored surface. The signature is stylized and appears to read 'M. Chanteloup'.

Déclaration sur l'honneur de non-plagiat

Je, soussigné(e),

Nom, prénom : Delaporte, Lucille

Élève ingénieur(e) régulièrement inscrit en 2^{me} année à TELECOM Nancy

Numéro de carte étudiante : 0710001189Y

Année universitaire : 2020/2021

Auteur, en collaboration avec Chanteloup Marie-Astrid, Graindorge Bleunwenn et Tilch Niels du rapport :

Projet Compilation - 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.

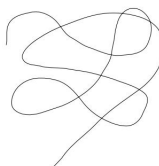
Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me 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 Nancy, le 17/04/2021

Signature :



Déclaration sur l'honneur de non-plagiat

Je, soussigné(e),

Nom, prénom : Graindorge Bleunwenn

Élève ingénieur(e) régulièrement inscrit en 2^{me} année à TELECOM Nancy

Numéro de carte étudiante : 31919591

Année universitaire : 2020/2021

Auteur, en collaboration avec Chanteloup Marie-Astrid, Delaporte Lucille et Tilch Niels du rapport :

Projet Compilation - 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 Nancy, le 17/04/2021

Signature :



Déclaration sur l'honneur de non-plagiat

Je, soussigné(e),

Nom, prénom : TILCH Niels

Élève ingénieur(e) régulièrement inscrit en 2^{me} année à TELECOM Nancy

Numéro de carte étudiante : 31916918

Année universitaire : 2020/2021

Auteur, en collaboration avec Chanteloup Marie-Astrid, Delaporte Lucille et Grain-dorge Bleunwenn du rapport :

Projet Compilation - 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 Nancy, le 17/04/2021

Signature :



Chapitre 2

Gestion de projet

2.1 Équipe de projet

L'équipe se compose de quatre étudiants en deuxième année :

- Marie-Astrid Chanteloup
- Lucille Delaporte
- Bleunwenn Graindorge
- Niels Tilch

Marie-Astrid est désignée cheffe de projet, elle aura la responsabilité d'animer les réunions et de suivre l'avancement du projet.

Bleunwenn est désignée secrétaire de projet, elle aura la responsabilité de rédiger les comptes-rendus de réunions et de mettre à jour le rapport de projet sur la plateforme GitLab.

2.2 Analyse du projet

2.2.1 Définition des objectifs

Les objectifs ont été définis à l'aide de la méthode SMART :

	Critère	Indicateur
S	Spécifique	L'objectif est défini clairement.
M	Mesurable	L'objectif est mesurable, par des indicateurs chiffrés ou livrables.
A	Atteignable	L'objectif doit être motivant sans être décourageant et doit apporter un plus par rapport au lancement du projet.
R	Réaliste	L'objectif doit être réaliste au regard des compétences et de l'investissement de l'équipe du projet.
T	Temporellement défini	L'objectif doit être inscrit dans le temps, avec une date de fin et des jalons.

2.2.2 Analyse des risques : Matrice SWOT

Nous avons évalué les risques liés au déroulement du projet mais aussi les qualités de l'équipe et les opportunités liées au projet. Ils sont résumés dans la matrice SWOT présentées en figure 2.1



FIGURE 2.1 – Matrice SWOT

2.3 Organisation du projet

Le projet se découpe en deux phases. La première prend place du mois de septembre 2020 jusqu'au début du mois de janvier 2021. La deuxième prend place du mois de janvier 2021 au mois de avril 2021.

Pour la deuxième partie du projet, nous avons découpé le projet en 21 étapes.

Elles ont été prévues dans le temps selon le diagramme de Gantt représenté en figure 2.2. Leur répartition entre les membres de l'équipe est présentée dans la matrice RACI en figure 2.3.

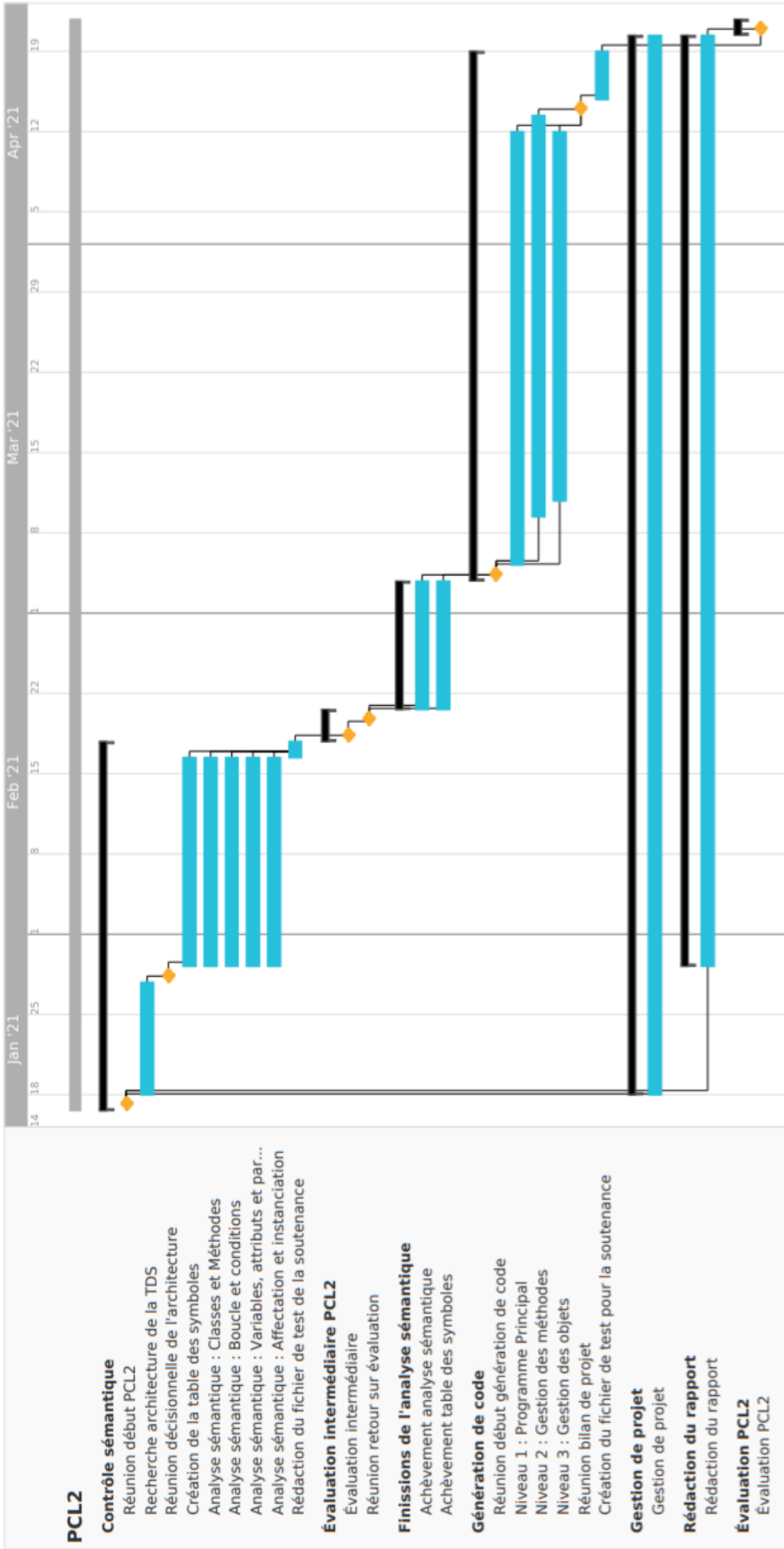


FIGURE 2.2 – Diagramme de Gantt

Matrice RACI								
		Niels	Marie-Astrid	Bleunwenn	Lucille			
Lots	Livrables ou taches PCL2							
Lot 1	TDS et Analyse Sémantique							
Lot 1.1	Implémentation de la structure de TDS							
Lot 1.1.1	Passage des éléments de Antlr à Java	C	C	R	R			
Lot 1.1.2	Ajout d'élément dans la TDS	R	R	R	R			
Lot 1.1.3	Ajout de TDS dans la TDS	R	R	R	R			
Lot 1.1.4	Affichage de TDS	A	A	A	R			
Lot 1.2	Analyse sémantique							
Lot 1.2.1	Analyse des classes	C	C	R	R			
Lot 1.2.2	Analyse des constructeurs	C	C	R	R			
Lot 1.2.3	Analyse des méthodes	C	C	R	R			
Lot 1.2.4	Analyse des blocs	C	R	A	A			
Lot 1.2.5	Analyse des opérations	C	A	R	C			
Lot 1.2.6	Analyse des paramètres attributs et variables	R	C	A	A			
Lot 1.3	Jeux de tests							
Lot 1.3.1	Affichage de TDS	C	C	C	R			
Lot 1.3.2	Test erreur sémantique	A	R	R	C			
Lot 2	Génération de code							
Lot 2.1	Niveau 1 - Programme Principal							
Lot 2.1.1	Création parcours génération de code	C	C	R	A			
Lot 2.1.2	Initialisation des fichiers de code	C	C	A	R			
Lot 2.1.3	Création d'environnement	C	C	R	A			
Lot 2.1.4	Obtention des variables et paramètres	R	A	C	R			
Lot 2.1.5	Affectation de valeurs	R	R	C	A			
Lot 2.1.6	Expression arithmétiques	C	C	C	R			
Lot 2.1.7	Structures de contrôles	C	R	C	A			
Lot 2.2	Niveau 2 - Méthodes et chaînes de caractères							
Lot 2.2.1	Gestion des chaînes de caractères	C	A	C	R			
Lot 2.2.2	Appels de méthodes	C	C	R	R			
Lot 2.2.3	Récupération des paramètres	R	A	C	R			
Lot 2.2.4	Valeur de retour	A	C	R	A			
Lot 2.3	Niveau 3 - Gestion des objets							
Lot 2.3.1	Descripteurs de classe	C	R	C	A			
Lot 2.3.2	Allocation de place dans le tas	C	R	C	A			
Lot 2.3.3	Gestion des attributs statiques	C	R	C	C			
Lot 2.3.4	Constructeur	C	R	A	C			
Lot 2.4	Jeux de tests							
Lot 2.4.1	Niveau 1 - Programme Principal	C	R	A	A			
Lot 2.4.2	Niveau 2 - Méthodes et chaînes de caractères	C	R	C	A			
Lot 2.4.3	Niveau 3 - Gestion des objets	C	R	C	C			
Lot 2.4.4	Final - Liste Chaînées	A	R	A	A			
Lot 3	Rapport							
Lot 3.1	GDP							
Lot 3.1.1	Compte-rendus de réunion	I	R	R	I			
Lot 3.1.2	Matrice RACI	I	R	I	I			
Lot 3.1.3	Diagramme de GANTT	R	A	I	I			
Lot 3.1.4	Charte de projet	I	A	I	R			
Lot 3.1.1	Bilan de groupe	I	R	R	I			
Lot 3.2	Rédaction et explication de la TDS							
Lot 3.2.1	Rédaction	I	I	R	A			
Lot 3.3	Rédaction et explication des tests sémantiques							
Lot 3.3.1	Rédaction	I	I	R	I			
Lot 3.4	Rédaction et explication de la génération de code							
Lot 3.4.1	Rédaction	I	R	R	I			
R	Réalisateur							
A	Approbateur							
C	Consultant							
I	Informé							

FIGURE 2.3 – Matrice RACI

2.4 Outils de travail

2.4.1 Réunion

L'équipe s'est réunie toutes les semaines de période scolaire le mercredi après-midi à des horaires variables, soit pour une réunion, soit pour session de travail collaboratif. Dans un premier temps, ces réunions ont eu lieu à distance grâce à la plateforme Discord, puis au sein du campus de TELECOM Nancy.

2.4.2 Partage du travail

Tout le long du projet, l'équipe a utilisé le répertoire GitLab fourni par l'école. Une branche `dev` a été créée pour l'écriture de la grammaire. Une branche `gdp` a été créée pour y consigner le rapport.

2.4.3 Rédaction du rapport

Le rapport a été rédigé sur ShareLatex pour permettre à tout les membres de compléter leurs éléments simultanément.

2.5 Comptes-rendu des réunions

2.5.1 Réunion du 15 janvier 2021

Présent	Absent	Heure	Lieu
Marie-Astrid Chanteloup Lucille Delaporte Bleunwenn Graindorge Niels Tilch	Personne	12h	Discord

Ordre du jour

- 1. Retour des professeurs sur la grammaire
- 2. Rapport PCL 1
- 3. Lancement PCL 2
 - Objectif
 - Transition d’Antlr vers Java

Retour des professeurs sur la grammaire

Un premier RdV a eu lieu avec M Oster pour vérifier certains aspects de la grammaire et faire quelques corrections. Le retour a été très positif. La grammaire ressemble à ce qui est attendu et permettra de commencer la seconde partie du projet sur de bonnes bases.

Un second RdV avec Mme Collin est en préparation pour valider la nouvelle grammaire et permettre de lancer la partie TDS et tests sémantiques.

Rapport PCL 1

Le rapport est quasiment fini, il reste uniquement des petites modifications et corrections à réaliser. Chacun est amené à relire les parties des autres et à faire une relecture complète du rapport si possible dans la soirée.

Marie-Astrid postera le rapport une fois qu’il sera complet.

Lancement de PCL 2

Objectif

Un rappel des objectifs et livrables de PCL 2 est fait :

- Création d’une TDS
- Jeu de tests sémantique et leurs rapports d’erreur
- Test sur ces éléments
- Génération de code
- Jeu de tests pour la génération de code
- Rapport de projet PCL 2

L’ensemble de ces éléments sont générés à partir de l’AST et des éléments créé par Antlr. Il est donc important de bien comprendre la transition de Antlr vers Java. La génération de code n’intervient qu’après la mise en place de la TDS et des tests sémantiques.

Transition d’Antlr ver Java

Lors de la compilation de la grammaire par AntlrWorks, deux fichiers sont produits. Il faut les prendre en main, pour pouvoir réfléchir à la mise en place de la TDS.

TO-DO LIST

- Rapport PCL 1 :
 - Relecture - Tous
 - Mise-en-ligne sur le Git - Marie-Astrid
- PCL 2 : Se renseigner sur le fonctionnement des fichiers créés par Antlr - Tous

Prochaine réunion : 27/01/2021

2.5.2 Réunion du 27 janvier 2021

Présent	Absent	Heure	Lieu
Marie-Astrid Chanteloup Lucille Delaporte Bleunwenn Graindorge Niels Tilch	Personne	14h	Discord

Ordre du jour

- 1. Fonctionnement des lexer et parser
- 2. Structure de TDS

Fonctionnement des lexer et parser

Avec l'aide de Tom Cabarat, Bleunwenn a réussi à générer des lexer et parser fonctionnels. Elle a aussi réalisé une lecture et création d'AST : le main lit un fichier de code Blood grâce au lexer et au parser puis parcourt l'AST créé grâce à la classe Parcours. Pour le moment, cette classe ne fait qu'une impression de l'arbre, mais à terme, il servira à faire le parcours de l'arbre pour la création de la TDS et l'analyse sémantique.

Structure de la TDS

Niels propose de créer une TDS par token. Lucille note que cela risque d'être très lourd et peu efficace et qu'il vaudrait mieux en envisager une par règle. Marie-Astrid rappelle que la TDS représente des structures et qu'il faudra trouver un entre-deux pour avoir un nombre suffisant de TDS et que celles-ci aient un sens.
Une première base de TDS est proposée où des attributs supplémentaires s'ajouteraient. La base contiendrait :

- un numéro de région
- un numéro d'imbrication
- l'adresse de la TDS mère (Null pour la TDS initiale)
- une liste de ses différents enfants (classe, méthode, constructeurs, attributs, etc) contenue dans une hashmap ordonnée, de pour lesquels la clé est le nom de l'élément

Il est proposé d'élaborer les TDS selon cette base avec les éléments supplémentaires selon la répartition suivante :

- Bleunwenn : classe, attributs et paramètres (attention à l'offset négatif/positif)
- Niels : méthode et constructeurs (discussion possible sur les paramètres avec Bleunwenn)
- Lucille : root, programme principal, envoi de message
- Marie-Astrid : Boucles conditionnelles (if et while)

TO-DO LIST

- Établissement des différentes TDS - Tous

Prochaine réunion : 03/02/2021

2.5.3 Réunion du 3 février 2021

Présent	Absent	Heure	Lieu
Marie-Astrid Chanteloup Lucille Delaporte Bleunwenn Graindorge Niels Tilch	Personne	14h	Discord

Ordre du jour

- 1. Re-répartition des éléments pour la construction de la TDS

Re-répartition des éléments pour la construction de la TDS

La répartition évoquée à la dernière réunion ne fonctionne pas bien à cause des multiples recouvrements des domaines.
Une nouvelle répartition est proposée en fonction des tokens. Tous les tokens n'auront pas un traitement spécifique, mais sont quand même répartis dans un esprit de vérification exhaustive. La répartition est notée ci-dessous :

Lucille

- ROOT
- BLOCPRINCIPAL
- BLOC
- BLOCCLASSE
- CLASSDEF
- EXTENDS
- CONSDEF
- METHODEDEF
- TYPERETOUR

Bleunwenn

- IF
- THEN
- ELSE
- WHILE

Niels

- CAST
- INST
- MESSAGE
- SELECT
- MESSOUSELECT

Marie-Astrid

- LISTFORMALPARAM
- FORMALPARAM
- LISTPARAM
- DECLVARIABLE
- PARAM
- ATTRIBDEF
- LISTPARAMAPPELCONSTR
- LISTOPERATEURS

- TO-DO LIST**
- Établissement des différentes TDS selon les tokens attribués - Tous

Prochaine réunion : 09/02/2021

2.5.4 Réunion du 9 février 2021

Présent	Absent	Heure	Lieu
Marie-Astrid Chanteloup Lucille Delaporte Bleunwenn Graindorge Niels Tilch	Personne	12h	Cafétéria de TELECOM Nancy

- Ordre du jour**
1. Retour sur la TDS
 2. Lancement des contrôles sémantiques

Retour sur la TDS

Un point est fait sur l’avancée de la TDS.
Les TDS des boucles conditionnelles, méthodes et blocs fonctionnent et généralement la TDS est bien générée.
Il reste quelques problèmes à régler :

- Marie-Astrid note que la gestion du mot-clef static n’est pas du tout prise en compte dans la structure Element. Lucille propose d’ajouter un attribut à la classe.
- DECLVARIABLE est réattribué à Niels
- L’instanciation est équivalente à un " :=" et n’est donc pas gérée dans la TDS
- De même pour le cast
- Le déplacement n’est pas géré pour le moment
- Les TDS de String et Integer et leurs fonctions de bases doivent être enregistrées dans la TDS

Généralement il est rappelé à tous de commenter son travail pour aider les autres à mieux le comprendre.

Lancement des contrôles sémantiques

L'équipe a un temps court pour réaliser cet élément plutôt long à faire car exhaustif. Il sera d'autant plus court que Niels et Marie-Astrid ont tous les deux des consolidations du S7 à passer et ne seront donc pas disponibles avant jeudi soir. Il faudra donc respecter ces deadlines pour que l'ensemble puisse être vérifié rapidement par les autres. De même, il sera important de bien tenir les autres au courant de son avancée et/ou ses difficultés.

Avant de passer à la répartition, un choix commun est fait quand à l'ordre du programme : l'ordre d'écriture est l'ordre de définition, ie une classe ne peut utiliser des éléments d'une classe définie ultérieurement.

- Répartition des éléments de contrôles sémantiques :
- Lucille et Bleunwenn en tandem :
 - Définition de classe
 - Définition de méthode
 - Si cela est possible, envoi de message et calculs
 - Niels :
 - Contrôle sémantique sur les noeuds : DECL_VAR, CAST, INST et :=
 - Marie-Astrid :
 - Bloc et bloc de classe
 - Rappel des éléments notés dans le rapport PCL1 sur les erreurs acceptées par la grammaire

TO-DO LIST

- Fin de l'établissement de la TDS - Tous
- Contrôles sémantiques à réaliser selon la répartition vue dans le point associé : Tous

Prochaine réunion : 17/02/2021

2.5.5 Réunion du 17 février 2021

Présent	Absent	Heure	Lieu
Marie-Astrid Chanteloup Lucille Delaporte Bleunwenn Graindorge Niels Tilch	Personne	15h	Salle de travail de TELECOM Nancy

Ordre du jour

1. Retour des professeurs sur la TDS et les contrôles sémantiques
2. Point communication au sein du groupe
3. Rapport PCL 2

Retour des professeurs sur la TDS et les contrôles sémantiques

La soutenance a eu lieu avec M Oster en début d'après-midi. Le retour est très positif. Une cinquantaine de contrôles sémantiques ont été présentés. Il reste encore à mettre en place les descripteurs de classe, ainsi que des contrôles sémantiques sur les envoies de messages.

Point communication et méthode de travail au sein du groupe

Certains aspects des méthodes de travail et de la communication du groupe sont abordés dans le but de les améliorer pour la fin du projet. Marie-Astrid rappelle que l'ensemble des comptes-rendus de réunion sont disponibles sur le drive du projet, et qu'il ne faut pas hésiter à les consulter pour retrouver la liste de travail à faire ou les décisions prises par le groupe. Pour ce qui est du code, Lucille rappelle qu'il faut toujours tester et commenter son travail avant de le déposer sur le Git. Il est également demandé aux membres du groupe de nettoyer le code après les tests. Si un membre du groupe découvre une erreur, il faut en informer l'équipe et expliciter au maximum le problème.

Rapport PCL 2

Marie-Astrid propose de commencer rapidement la rédaction du rapport sur la conception de la TDS et les contrôles sémantiques afin de ne pas oublier d'informations. Chacun devra aussi compléter le rapport d'étonnement de mi-parcours. La semaine de vacances à venir sera consacrée à la rédaction, la génération de code sera évoquée à la rentrée.

TO-DO LIST

- Terminer de corriger les derniers problèmes des contrôles sémantiques - Tous
- Rapport PCL 2
 - Rédaction du rapport d'étonnement - Tous
 - Rédaction de la partie sur la TDS - Bleunwenn
 - Rédaction de la sous-partie sur la classe Element - Lucille
 - Rédaction de la partie sur les contrôles sémantiques - Tous

Prochaine réunion : 03/03/2021

2.5.6 Réunion du 3 mars 2021

Présent	Absent	Heure	Lieu
Marie-Astrid Chanteloup Lucille Delaporte Bleunwenn Graindorge Niels Tilch	Personne	14h	Cafétéria de TELECOM Nancy

Ordre du jour

1. Retour sur le rapport
2. Lancement de la génération de code

Retour sur le rapport

Un point est fait sur l'avancée de la rédaction du rapport. La partie sur la classe Element est presque terminée. L'organisation des parties TDS et Contrôles sémantiques est faites, et les images illustrant la TDS ont été ajoutées. La partie sur les contrôles sémantiques n'est pas encore commencée car il faut se mettre d'accord sur la forme à lui donner. Le groupe décide de réaliser un listing des erreurs sémantiques traitées, mais d'organiser les contrôles en parties et sous-parties commentées. Les personnes n'ayant pas complété leur rapport d'étonnement doivent le faire dans la soirée. Il faudra également ajouter les nouveaux éléments de gestion de projet au rapport.

Lancement de la génération de code

Pour commencer la génération de code, il faut utiliser un nouveau **Filewriter** dans lequel apparaîtront les commandes en langage assembleur. Le document généré sera à tester sur MicroPuipk sur de tout petits exemples pour commencer. Afin de ne pas compiler en présence d'erreurs sémantiques, il faut un nouveau parcours d'arbre, qui sera réalisé uniquement lorsque le fichier d'erreurs sémantiques sera vide.

TO-DO LIST

- Terminer la rédaction du rapport - Bleunwenn
- Réaliser le nouveau parcours d'arbre - Bleunwenn
- Créer les descripteurs de classe à ajouter à la TDS - Lucille
- Gestion des déplacements dans la pile - Marie-Astrid
- Gérer les protocoles d'appel de fonction - Niels

Prochaine réunion : 21/03/2021

2.5.7 Réunion du 17 mars 2021

Présent	Absent	Heure	Lieu
Marie-Astrid Chanteloup Lucille Delaporte Bleunwenn Graindorge Niels Tilch	Personne	14h	Cafétéria de TELECOM Nancy

Ordre du jour

1. Point de route
2. Suite du projet
 - Retour sur les deadlines
 - Étude fiche de niveaux

Point de route

Chaque membre du groupe fait un point sur ce qu’il a réalisé depuis la dernière réunion. Bleunwenn a déposé sur le Git le nouveau fichier de parcours de l’AST pour la génération de code. Le rapport a également avancé, la partie sur la TDS est quasiment complète. Lucille a terminé les descripteurs de classe et à complété le Main pour que la génération de code ne se fasse qu’en l’absence d’erreurs sémantiques. Niels a rédigé son rapport d’étonnement et a terminé le Gantt de la deuxième partie du projet. Marie-Astrid a rédigé les comptes-rendus des dernières réunions sur le rapport et a terminé la gestion des offsets.

Suite du projet

Retour sur les deadlines

Les soutenances finales du projet ont lieu les 20 et 21 avril et le rapport est à rendre le vendredi 16 avril à 12h. Le groupe décide de se fixer le mercredi 14 avril comme date de fin de code. Cela permettra d’éviter les erreurs de dernières minutes et de réserver la fin de la semaine à la rédaction du rapport et à sa relecture. Lors de la génération de code, il faudra penser à déposer ses fichiers de tests intermédiaires sur le Git pour pouvoir facilement les utiliser et les regrouper avant la soutenance.

Etude fiche de niveaux

Mme Collin a envoyé par mail une fiche récapitulative des niveaux d’implémentation et d’évaluation. Il y a quatres niveaux, et les deux premiers sont a réaliser pour obtenir la note moyenne. Cette fiche va permettre de répartir le travail et les objectifs de la semaine. Pour la semaine prochaine, il serait bien d’avoir terminé les attendus du premier niveau. Les trois élèves du groupe IL ont étudié de nombreux éléments utiles à la PCL 2 dans leur cours de Traduction des Langages. Marie-Astrid fournira une copie de ses notes de cours à Bleunwenn afin qu’elle puisse être à jour sur le projet.

TO-DO LIST

- Génération de code des opérations - Lucille
- Génération de code des boucles conditionnelles - Marie-Astrid
- Génération de code des déclarations de variables - Bleunwenn
- Génération de code de la récupération de paramètres et des affectations - Niels
- Partager les cours de TRAD2 à Bleunwenn - Marie-Astrid

Prochaine réunion : 24/03/2021

2.5.8 Réunion du 24 mars 2021

Présent	Absent	Heure	Lieu
Marie-Astrid Chanteloup Lucille Delaporte Bleunwenn Graindorge Niels Tilch	Personne	15h	Cafétéria de TELECOM Nancy

Ordre du jour

1. Point d’avancement
2. Retour sur les objectifs

Point de route

Lucille a terminé de générer le code de préparation d’un programme : les registres 13, 14 et 15 sont réservés pour les pointeurs de pile. Elle a également terminé de traiter les opérations sur les entiers purs. Il reste à régler le problème des Strings et des variables. Marie-Astrid a terminé la génération de code sur les boucles conditionnelles, il ne reste plus qu’à tester l’ensemble sur MicroPuipk. Bleunwenn a commencé la génération de code des déclarations de variables. Niels a commencé a travailler sur la récupération de variables et a constaté que cela serait proche de la gestion des paramètres, de la récupération des attributs et des méthodes.

Retour sur les objectifs

Le groupe constate que le niveau 2 n’est pas testable tant que le niveau 3 n’a pas été implémenté. Il faudra donc au moins aller jusqu’au niveau 3 pour s’assurer une note correcte. De plus, la façon dont la TDS a été construite permet de sauter quelques étapes des niveaux 1 et 2, et de les réaliser en même temps que celles du niveau 3. L’allocation mémoire des déclarations de variables ne sera pas nécessaire car elle sera gérer directement avec le déplacement maximal obtenu pour chaque bloc du programme.

TO-DO LIST

- Opérations arithmétiques sur les String - Lucille
- Gestion de la méthode print - Lucille
- Déclaration et environnement de bloc - Bleunwenn
- Finir les recherches de paramètres et variables - Niels
- Gestion des affectations - Niels
- Terminer les tests sur les boucles - Marie-Astrid
- Finir les descripteurs de classe - Marie-Astrid
- Génération de code des méthodes - Lucille et Bleunwenn
- Gérer les instanciations - Niels si le travail d'avant n'est pas trop long

Prochaine séance de travail : 31/03/2021

2.5.9 Réunion du 7 avril 2021

Présent	Absent	Heure	Lieu
Marie-Astrid Chanteloup Lucille Delaporte Bleunwenn Graindorge Niels Tilch	Personne	14h	Cafétéria de TELECOM Nancy

Ordre du jour

1. Point d'avancement
2. Suite du projet

Point de route

Lucille a terminé de générer le code pour la concaténation de String, l'utilisation de la méthode `print` et les allocations de Strings dans le tas. L'ensemble fonctionne pour toute sorte d'entrées, même avec des valeurs ASCII. La gestion des opérations est quasiment terminée. Il ne manque que les variables pour lesquelles il faut attendre d'avoir la fonction de récupération. Suite à une incompréhension de l'obtention des variables avec Niels, cette fonction n'est pas encore opérationnelle.

Niels travail actuellement à la correction de la recherche de variables, et à la récupération de leur adresse. Les affectations ne sont pas encore terminées, il faut encore tester le programme. Il devrait en théorie fonctionner sur les envoies de messages. Une question sur l'affectation des Strings est posée. Lucille répond que cela est entièrement géré par les opérations récursives.

Bleunwenn a avancé sur le rapport pendant la semaine. L'ensemble des comptes-rendus de réunion ont été mis en page sur le support LaTeX, et les deux parties sur la table des symboles et l'analyse sémantique sont quasiment terminées. Il restera la partie sur la génération de code à ajouter avant le rendu. La génération de code pour les méthodes a également été commencée. Une séance de travail sera prévue avec Lucille à la fin de la semaine pour terminer cette partie.

Marie-Astrid a terminé les descripteurs de classe. Elle a également travaillé sur l'allocation dans le tas pour les instanciations de classe, le code sera déposé sur Git dans la soirée. La génération de code sur les boucles a été simplifiée et Marie-Astrid a commencé la gestion des `statics`. Il lui manque le code de la récupération des attributs, traité par Niels, pour terminer son travail.

Suite du projet

Il ne reste qu'une semaine avant le rendu du rapport. L'équipe veut terminé le niveau 2 avant mercredi 14 pour se concentrer sur le rapport à la fin de la semaine. Il est demandé à chaque membre de compléter rapidement le bilan personnel et le bilan d'équipe du projet dans le rapport, et de commencer à penser au contenu de la partie génération de code.

TO-DO LIST

- Terminer les opérations avec les variables - Lucille
- Gestion des méthodes println, itoa (`Int to String`) - Lucille
- Génération de code sur les méthodes - Bleunwenn
- Rédiger les rapports de réunions - Bleunwenn
- Finir les affectations - Niels
- Terminer le Gantt de la deuxième partie du projet - Niels
- Gérer la récupération d'attributs - Marie-Astrid
- Faire la matrice SWOT de la deuxième partie du projet - Marie-Astrid
- Préparer le bilan des tâches réalisées - Marie-Astrid
- Préparer les bilans personnels, bilans d'équipes et bilan horaires - Tous

Prochaine réunion : 14/04/2021

2.5.10 Réunion du 14 avril 2021

Présent	Absent	Heure	Lieu
Marie-Astrid Chanteloup Lucille Delaporte Bleunwenn Graindorge Niels Tilch	Personne	16h	Cafétéria de TELECOM Nancy

Ordre du jour

1. Point sur le code
2. Point sur le rapport
3. Préparation et rédaction du bilan
4. Fin de projet

Point sur le code

Niels a terminé de générer le code pour les paramètres et les affectations. L’ensemble a été testé plusieurs fois et déposé sur le Git. Lucille a terminé les opérations sur les Strings. Il reste encore des erreurs de registres à corriger, et la gestion des sélections dans les affectations. Lucille conseille d’utiliser directement le travail réalisé sur les messages ou selections afin d’éviter de refaire un nouveau cas. Au niveau de la gestion des étiquettes, il faut modifier le nom des étiquettes des méthodes pour éviter les confusions lorsque plusieurs classes ont des méthodes de même nom. Au final, les niveaux 1 et 2 de la fiche des niveaux fournie par Mme Collin sont validés. Pour le niveau 3, la majorité des points sont gérés. Les descripteurs de classes sont créés et fonctionnels mais ne sont pas encore utilisés dans la génération de code, l’équipe ayant réussi à utiliser les étiquettes pour le moment.

Point sur le rapport

Dans la gestion de projet, Marie-Astrid a terminé la matrice SWOT et Bleunwenn a terminé la rédaction des comptes-rendus de réunion. Niels doit terminer le Gantt rapidement afin que Marie-Astrid puisse réaliser la matrice RACI. La rédaction des parties du rapport sur la TDS et l’analyse sémantique est terminée, il ne reste qu’à les relire et à vérifier la mise en page. Pour la partie sur la génération de code, il est demandé aux membres du groupe de noter les idées principales de ce qu’ils ont codé afin de ne rien oublier lors de la rédaction.

Préparation et rédaction du bilan

Les bilans personnels sur la deuxième partie du projet doivent être remplis par chaque membre du groupe. Marie-Astrid va compléter le tableau bilan du travail attendu et réaliser la table horaire. Il faudra que chaque membre du groupe indique le nombre d’heures passées sur les éléments du projet. Le bilan global du projet réalisé par l’ensemble du groupe se trouve dans la partie Bilan du rapport.

Fin de projet

Voici les éléments restant pour être prêts pour le rendu du projet :

TO-DO LIST

- Réaliser le fichier de tests pour la soutenance - Lucille et Niels
- Noter les idées principales de la génération de code - Tous
- Réaliser le Gantt - Niels
- Réaliser la matrice RACI - Marie-Astrid
- Rédiger les bilans personnels et le bilan horaires - Tous
- Rédiger le compte-rendu de réunion et le bilan global - Bleunwenn
- Relire le rapport - Tous

Date de fin de projet : 19/04/2021

Chapitre 3

Table des Symboles

3.1 Organisation de la TDS

3.1.1 Définition de la TDS

La table des symboles permet de centraliser les informations liées aux identificateurs des programmes écrit en langage Blood. Elle sauvegarde, pour chaque bloc, l'ensemble des variables, paramètres et procédures avec leur nature, leur type, leur type de retour (si il en existe un), leur emplacement mémoire grâce aux déplacements, leur numéro d'imbrication et de région.

Chaque TDS est caractérisée par son numéro de région et son numéro d'imbrication. On y retrouve le numéro de sa TDS parente ainsi que des informations différentes selon le type d'élément ajouté dans la table.

Nous avons créé une TDS de base, comportant le nombre de régions du programme puis les classes et méthodes déjà prédéfinies par notre langage. Les classes initialement définies par notre langage (`Integer` et `String`) se trouvent dans les TDS 2,1 et 3,1, et sont suivies des méthodes classiques sur ces types, comme `println()`, `print` et `toString`. Voici un aperçu de ces TDS préexistantes :

```
----- TDS 0, 0 -----
/-/ Tds Initiale /-/
class : String      numRégion : 1
class : Integer     numRégion : 2
class : UneClasse   numRégion : 7
class : Couleur     extends : UneClasse      numRégion : 10
numRégion du main : 14
----- TDS 1, 1 -----
/-/ String /-/
methode : println   typeRetour :      numRégion : 3
methode : print     typeRetour :      numRégion : 4
----- TDS 3, 1 -----
/-/ println /-/
----- TDS 4, 1 -----
/-/ print /-/
----- TDS 2, 1 -----
/-/ Integer /-/
methode : toString  typeRetour :      numRégion : 5
----- TDS 5, 1 -----
/-/ toString /-/
```

Nous avons dans un premier temps créé une classe TDS permettant de construire les tables de symboles selon la nature de l'élément rencontré.

3.1.2 Éléments présents dans une TDS

Sachant que la classe TDS était commune aux TDS de tous les blocs du programme, nous avons alors créé une classe `Element` commune à tous les types de TDS. Les différents types d'éléments sont listés ci-dessous et possèdent chacun, pour les décrire, un constructeur de la classe `Element`.

Les variables et les attributs Toutes les TDS - sauf la TDS 0, 0 - peuvent posséder des variables, attributs ou paramètres. Ces trois éléments sont décrits de la même manière par le constructeur de `Element` associé. La différence entre les attributs et les paramètres/variables, est leur position dans les TDS. Un attribut est situé dans un bloc de classe ou une bloc principal, sinon c'est une variable ou un paramètre. Une variable se différencie, quant à elle, d'un paramètre par son déplacement qui est positif (contrairement à celui du paramètre). Ce type d'élément est décrit par 5 attributs : un boolean pour savoir si l'attribut est statique : `isStatic`, son `typeRetour`, un boolean qui décrit si son type est primitif ou non : `isPrim`, son nom : `name` et son `deplacement`.

Les méthodes Seules les TDS de classe peuvent posséder un élément méthode. Ce type d'élément est défini par 5 attributs : un boolean pour savoir si la méthode est statique : `isStatic`, son `typeRetour`, un boolean qui décrit si la méthode override une autre méthode d'une classe parente : `isOverride`, son nom : `name` et son `numRegion`.

3.1.3 Traitement et affichage de ces éléments

Affichage d'une classe dans une TDS Les noms des classes présentes dans le programme sont affichés dans la TDS 0,0 avec les classes prédéfinies. On y affiche leur nom ainsi que le nom de la classe qu'elle étend s'il y en a une. Le numéro de région de leur TDS fille mène à la TDS de la classe proprement dit, contenant l'ensemble des éléments définis dans le bloc de cette classe. Dans l'image ci-dessous, nous pouvons voir la TDS de la classe `UneClasse`, avec ses attributs, son constructeurs et les méthodes qui y sont définies.

```
----- TDS 7, 1 -----
/-/ UneClasse /-/
PERE : 0.0
type : Integer      nom : jeSuisUnAttribut      offset : 6
static type : String  nom : etMoiUnAutre      offset : 0
constructeur : UneClasse      numRégion : 8
methode : methode      typeRetour : null      numRégion : 9
```

Affichage d'un attribut et d'une variable dans une TDS Pour les attributs et les variables, la TDS affichera son type et son nom. Elle comportera également son offset, permettant de gérer plus facilement les déplacements lors de la génération de code. L'offset est réalisé de manière cumulative dans notre parcours. Pour les attributs, on place initialement le pointeur de pile en dessous de l'adresse de retour, et les offsets sont négatifs. Pour les variables en revanche, on commence à pointer au dessus des chaînages statique et dynamique, et les offsets sont positifs.

Gestion des conditions dans la TDS Les boucles `while` et `if` sont retrouvables dans les TDS. Elles apparaissent dans la TDS du bloc dans lequel elles ont été appelées. Leur mention est suivi du numéro de région de la TDS associée à la condition. Pour les boucles `while`, il n'y a qu'une TDS associée. Pour les conditions `if`, on associe deux TDS : une première pour le bloc `THEN` de la condition, et une seconde pour le bloc `ELSE`. Pour les deux types de boucle, on retrouvera l'ensemble des éléments définis à l'intérieur des blocs conditionnels dans la TDS.

```
----- TDS 7, 1 -----
/-/ main /-/
PERE : 0.0
IF_BLOC :      THEN : 8.2      ELSE : 9.2
IF_BLOC :      THEN : 10.2     ELSE : 11.2
WHILE_BLOC : 12.2
WHILE_BLOC : 13.2
```


Affichage des constructeurs et des méthodes dans une TDS Le constructeur apparaît dans la TDS de sa classe parente avec son nom, correspondant au nom de sa classe, ainsi que la classe dont il hérite. Il est suivi de son numéro de région, permettant de retrouver sa propre TDS. Dans sa TDS on retrouve les paramètres du constructeur, définis par leur types, leur nom et leur offset. Pour une méthode, la TDS de la classe parente va contenir le nom de la méthode, son type de retour (de valeur null si elle n'en a pas), et les indicateurs `override` et `static` si ils se trouvent dans la définition. Grâce au numéro de région, on accède à la TDS de la méthode qui contient l'ensemble des attributs et instructions définis dans le bloc de la méthode. L'image ci-dessous représente la TDS de la classe Couleur (TDS 10,1), ainsi que deux de ses TDS filles, associées au constructeur de la classe et à la méthode `methode`.

```
----- TDS 10, 1 -----
/-/ Couleur /-/
PERE : 0.0
type : Integer    nom : arg1    offset : -2
type : String     nom : arg2    offset : -4
constructeur : Couleur extends : UneClasse numRégion : 11
override methode : methode typeRetour : null numRégion : 12
static override methode : uneMethode typeRetour : null numRégion : 13
----- TDS 11, 2 -----
/-/ Couleur /-/
PERE : 10.1
type : Integer    nom : arg1    offset : -2
type : String     nom : arg2    offset : -4
----- TDS 12, 2 -----
/-/ methode /-/
PERE : 10.1
type : Integer    nom : arg     offset : -2
type : String     nom : argum   offset : -4
type : Integer    nom : unAutreAttribut offset : 6
```


Chapitre 4

Contrôle Sémantique

L'analyse sémantique a lieu après les analyses lexicales et syntaxiques mais avant la génération de code. Elle permet de signaler à la compilation, les erreurs du code qui ne sont pas détectées par la grammaire, comme des erreurs de typages statiques, ou de définition. Dans notre projet, nous avons choisi de ne réaliser qu'une seule passe, c'est pourquoi cette analyse a lieu lors de la création de la table des symboles. L'ensemble des erreurs sémantiques trouvées sont inscrites dans un nouveau fichier généré lors de notre parcours d'arbre. Le numéro de la ligne à laquelle se trouve l'erreur est affiché pour en faciliter la correction. Au total, une quarantaine de tests ont été implémentés.

4.1 Analyse sémantique des classes

Les tests sémantiques sur les classes permettent notamment de vérifier qu'une classe est correctement définie (paramètres, héritage...). Voici quelques tests que nous avons mis en place sur nos classes :

4.1.1 Déclaration

- Vérifier que la classe n'est pas déjà définie
- Vérifier que plusieurs paramètres de la classe n'ont pas le même nom
- Si il y a héritage, vérifier que la super classe existe
- S'assurer que l'héritage en boucle est impossible

4.1.2 Gestion du *this*

L'identificateur réservé `this` a le même sens qu'en Java. On réalise plusieurs tests sur cet identificateur :

- Vérifier que l'identificateur est bien dans une classe
- Vérifier que la variable sur lequel est appelé le mot clé est bien un attribut de la classe

4.2 Analyse sémantique des paramètres, attributs et variables

Les différents tests sur les attributs, les paramètres et les variables sont souvent très proches. Nous avons réaliser des tests sur leur bonne déclaration et utilisation dans le programme.

4.2.1 Déclarations

- Lors des déclarations de paramètres, variables ou attributs, vérifier que le type utilisé existe bien
- Pour les variables, vérifier que deux variables ne sont pas définies de la même façon dans une même TDS

4.2.2 Instanciation

- Lorsque la déclaration de variable a lieu en même temps que l'instanciation, s'assurer que le type instancié est bien le même que celui de la classe
- Lorsqu'il n'y a pas de déclaration de variable, vérifier que la variable a bien été déclarée avant dans un bloc correct

4.2.3 Affectations

Les principales vérifications à effectuer sont celles sur la compatibilité des types. Dans une affectation de la forme `a := 1;`, on s'assure que le type du noeud de gauche correspond bien à celui du noeud de droite.

4.3 Analyse sémantique des méthodes

4.3.1 Constructeur

Dans l'ensemble, les tests sémantiques du constructeur ressemblent beaucoup à ceux de la classe. Il faut vérifier la bonne définition du constructeur, de ses paramètres et la bonne gestion de l'héritage.

- Comme pour la classe, vérifier que plusieurs paramètres du constructeur n'ont pas le même nom
- Vérifier que le nom du constructeur est le même que celui de sa classe
- Vérifier que les paramètres du constructeur et de la classe sont identiques (par exemple, lorsque le paramètre de la classe est précédé d'un **var**, celui du constructeur doit également être précédé d'un **var**)
- Si la classe du constructeur hérite d'une autre classe, s'assurer que le constructeur hérite également de la super classe et que son **extend** est le même que celui de sa classe. Il faut alors également respecter les appels de paramètres de la super classe.
- S'assurer que le constructeur ne fait pas d'appel à une super classe si sa classe ne l'extend pas

4.3.2 Définition de méthode

- Vérifier que plusieurs paramètres n'ont pas le même nom
- Vérifier que les paramètres n'ont le même nom que les attributs de la classe
- Vérifier que le nom de la méthode ne correspond pas à un mot interdit (mot utilisé d'une autre façon dans la grammaire, tel que **is**, **override**, **then**...). Ce travail est normalement déjà réalisé avec la grammaire.
- Vérifier que le type de retour de la méthode existe bien
- Vérifier que le nom de la méthode n'est pas utilisé deux fois dans la classe
- Vérifier la bonne gestion des overrides : vérifier l'existence de la méthode dans une autre classe quand elle override, s'assurer qu'une méthode n'override que lorsqu'il y a héritage pour sa classe...

4.4 Analyse sémantique des blocs

Un bloc est défini par des accolades et peut comprendre une liste vide d'instruction ou une liste de déclarations de variables locales, suivies du mot clé **is** et d'instructions. Il est nécessaire de contrôler la bonne utilisation des mots clés **is** et **result** dans notre analyse sémantique.

4.4.1 Gestion du **is**

Le mot **is** est un mot clé réservé, utilisé dans les blocs de déclaration de classe et de méthode. Il est suivi de déclarations d'attributs, des méthodes et du constructeur de la classe dans un ordre quelconque. Sur ce mot clé, on réalise les tests suivants :

- Vérifier la présence des mots clés **is** ou **result**
- En cas de présence d'un **is**, s'assurer de la présence d'au moins une déclaration de variable et de l'absence d'instruction avant le mot clé
- En cas de présence d'un **is**, s'assurer de la présence d'au moins une instruction et de l'absence de déclaration de variable après le mot clé

4.4.2 Gestion du **result**

Le **result** est une pseudo-variable utilisée pour les méthodes ayant un type de retour. Il permet de retourner le résultat de méthodes ayant un corps complexe. C'est un mot clé réservé, qui ne peut être utilisé que dans ces conditions.

- Vérifier que le mot clé **result** n'est présent que pour des méthodes avec des types de retour

4.5 Analyse sémantique des opérations

Les contrôles sur les opérations et la concaténation doivent se faire de manière récursive, afin de pouvoir réaliser plusieurs opérations à la fois.

- Vérifier que la division par 0 est impossible
- Vérifier que les opérations '+', '-', '*', et '/' ont bien lieu sur des Integers, sur les fils gauches et les fils droits
- Vérifier que l'opération de concaténation '&' a bien lieu sur des Strings, sur son fils gauche et son fils droit
- Cela implique donc de s'assurer que les opérations sur les Strings et les Integers ne sont pas mélangées

Chapitre 5

Génération de code

Il s'agit de la dernière étape de notre projet, celle permettant, à partir d'un code lexicalement, syntaxiquement et sémantiquement correct, de générer du code machine qui effectuera les instructions précédemment écrites dans le langage source. La génération de code se fait à l'aide d'un parcours d'arbre abstrait et des TDS, à la suite de l'analyse sémantique. Le nouveau fichier `GenerationCode.asm`, contenant le code assembleur, n'est généré que lorsque le fichier d'erreurs sémantiques est vide.

5.1 Principe de la génération de code

La génération de code est l'étape de la compilation qui permet de transformer un arbre syntaxique abstrait, sur lequel des tests sémantiques ont déjà été effectués, en code machine (assembleur) ou en code intermédiaire. Dans cette partie, afin de rester dans le cadre du projet, nous ne nous intéressons qu'à la génération de code machine.

Il s'agit donc de parcourir un AST et de le traduire en assembleur compréhensible par le processeur de la machine. Ce code assembleur, une fois compilé, pourra être exécuté et effectuera les instructions initialement écrites, dans notre cas, en langage Blood.

Les instructions assembleur correspondent à des instructions élémentaires effectuables par le processeur. Ces instructions permettent de faire des opérations telles que de la lecture ou de l'écriture en mémoire ou encore des opérations arithmétiques élémentaires. Afin de réaliser ces opérations élémentaires, le processeur utilise des registres, correspondants à des emplacements mémoire internes au processeur.

5.2 Conventions et choix dans notre gestion du générateur

La génération du code assembleur s'appuie sur un parcours de notre AST et de nos TDS en Java et génère des instructions assembleur microPIUPK dans un fichier `.asm`. Cet assembleur supporte 16 registres, numérotés de 0 à 15.

5.2.1 Conventions sur les pointeurs, exceptions et adresses

La fonction `initialisationCodeAssembleur()` définie dans la classe `CodeAssembleur` est utilisée pour réserver les registres, définir les numéros d'exceptions et initialiser la pile comme cela :

- **SP (Stack Pointer)** : pointeur de pile. L'adresse de la dernière valeur empilée est stockée dans le registre. Le registre utilisé est R15.
- **BP (Base Pointer)** : pointeur sur la base de l'environnement de la fonction courante. Il est placé dans le registre R14.
- **HP (Heap Pointer)** : pointeur sur le tas, l'adresse du tas est stockée dans un registre. Le registre utilisé est R13.
- **EXIT_EXC** : numéro d'exception de EXIT. Il s'agit du numéro permettant l'exécution d'une trappe logiciel indiquant la fin du programme. Sa valeur est fixée à 64.
- **READ_EXC** : représente le numéro d'exception du READ, utile pour l'exécution d'une trappe logiciel permettant la lecture d'une entrée utilisateur sur l'entrée standard. Sa valeur est fixée à 65.
- **WRITE_EXC** représente le numéro d'exception du WRITE, utilisé pour l'exécution d'une trappe logiciel permettant l'écriture d'une sortie sur la sortie standard. Sa valeur est fixée à 66.
- **NUL** est le caractère qui doit terminer une chaîne de caractères, sa valeur est fixée à 0.
- **NULL** correspond au pointeur nul. Sa valeur est fixée à 0.
- **NIL** est l'identificateur de fin de liste chaînée, sa valeur est fixée à 0.
- **INT_SIZE** représente la taille des entiers, sa valeur est fixée à 2.
- **ASCII_0** représente l'initialisation du premier caractère ASCII. Cette valeur est fixée à 30.
- **DESC_ADRS** correspond à l'adresse de base des descripteurs de classe. Elle est fixée à 0x0000 dans notre générateur.
- **HEAP_ADRS** représente l'adresse de base du tas, et est fixée à 0x4000.
- **STACK_ADRS** représente l'adresse de base de pile, sa valeur est fixée à 0xC000.

- **STATIC_ADRS** correspond à l'adresse de base des statiques et est située en bas de pile, à l'adresse 0xC000.
- **LOAD_ADRS** correspond à l'adresse de chargement du programme. Elle est fixée à 0xD000.

5.2.2 Convention sur le chaînage

Chaînage dynamique : ce chaînage est initialisé à NULL pour le main et au Base Pointer de l'environnement appelant pour les autres environnements.

Chaînage statique : ce chaînage est initialisé à NULL pour le main, au Base Pointer de l'environnement appelant pour les blocs **then**, **else** et **do**, et à l'adresse de l'instance de l'objet pour les méthodes.

5.2.3 Préparation du *Main*

Le main est lancé lorsque l'on rencontre un token BLOCPRINCIPAL dans notre AST. Lors de sa préparation, on stocke la valeur de la **STACK_ADRS** dans le pointeur SP et la valeur de la **HEAP_ADRS** dans le pointeur HP avant de réaliser un saut long vers le main.

5.3 Implémentation du niveau 1 : programme principal

5.3.1 Déclarations de variables

Lors de la création de notre TDS, nous avons choisi de récupérer les offset de façon cumulée. Pour chaque nouvelle variable, l'offset est incrémenté de la taille de cette variable. Grâce à ce fonctionnement, l'espace dans notre pile peut être réservé directement lors de la déclaration de classe ou méthode.

5.3.2 Récupération de variables

Il est possible de récupérer l'adresse d'une variable grâce à la fonction qui parcourt l'ensemble des éléments des TDS à la recherche du nom de la variable afin de récupérer son offset. On peut alors effectuer le déplacement pour obtenir la bonne adresse. Si l'élément n'est pas trouvé dans la TDS, on saute dans le chaînage statique.

De la même façon, la fonction `rechercheValeurVariableAssembleur()` permet de récupérer la valeur d'une variable. La valeur contenue dans la case de la variable est stockée dans un registre afin de pouvoir être récupérée ultérieurement.

Les sélections

Lors d'une sélection de la forme `a.attribut`, on récupère la tds de la classe de l'objet sur lequel s'effectue la sélection (objet `a` dans ce cas). On récupère également le déplacement de l'attribut et on ajoute ce déplacement à l'adresse de l'objet.

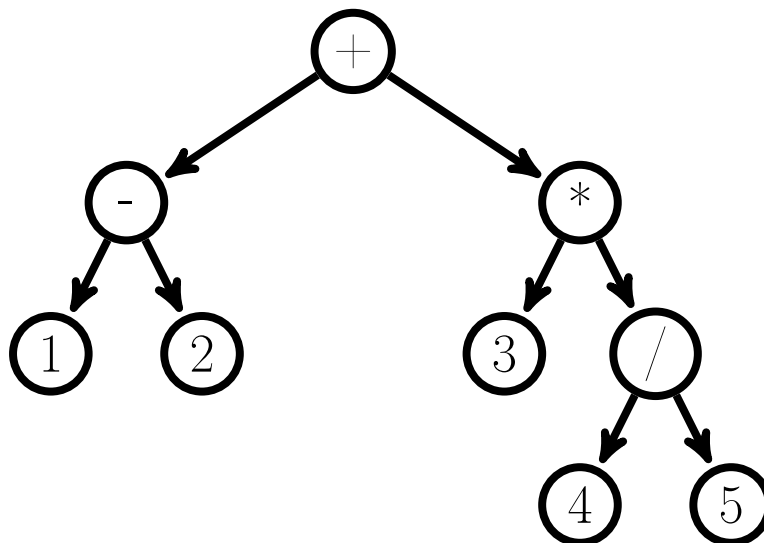
5.3.3 Affectations simples et affectations d'expressions arithmétiques

Afin d'effectuer une affectation, on récupère l'adresse de la variable de gauche. Cette adresse est stockée dans un registre. On stocke ensuite la valeur de l'élément de droite dans un second registre. Pour récupérer cette valeur, on peut faire appel à l'ensemble des fonctions de gestion de variables : les opérations et expressions arithmétiques ou encore les envoies de messages. On enregistre finalement la valeur récupérée dans l'adresse de la variable contenue en registre.

5.3.4 Expressions arithmétiques

La génération de code des opérations est gérée par une fonction récursive. Cette fonction gère les opérations avec des Integer positifs mais aussi avec des Integers négatifs et des Strings. Elle ne gère cependant pas du tout les opérateurs de comparaison.

La fonction est appelée lorsqu'un des opérateurs `+`, `-`, `*`, `/` ou `&` est rencontré dans l'arbre abstrait. Le cas terminal est effectué lorsque l'on rencontre des feuilles. Pour tester cela, on utilise la fonction `getChildCount()` qui nous permet de vérifier que le noeud n'a pas d'enfant. Lorsque les 2 noeuds rencontrés sont des feuilles, on enregistre chacune des valeurs dans un registre. On remonte ensuite au noeud père pour déterminer l'opération à effectuer, et on stocke le résultat. Si seulement l'un des noeuds rencontré est une feuille, on stocke sa valeur dans un registre et on appelle la fonction sur le deuxième noeud. Voici un exemple d'arbre sur la suite d'opérations `1-2+(1*(4/5))` :



Opérations sur les entiers

Dans le cas des additions, soustractions et multiplications, seuls deux registres sont nécessaires pour stocker les valeurs et effectuer l'opération. Pour les divisions, il est nécessaire d'utiliser trois registres pour effectuer les calculs. Cependant, dans chaque cas, seul un registre est finalement bloqué pour enregistrer le résultat. On attend ensuite la prochaine itération de la méthode pour soit libérer le registre (et en utiliser un autre qui était bloqué auparavant) soit si l'appel à la méthode est fini, on utilise le résultat de l'opération donc on libère aussi le registre.

Opérations sur les Strings

La seule opération pouvant être réalisée sur les chaînes de caractères est la concaténation avec l'opérateur '&'. Les chaînes de caractères sont enregistrées grâce à la méthode `recopieString()`, qui ajoute également une étiquette contenant le numéro d'apparition de la chaîne dans le code.

La méthode `allocationStringTas()` utilisée dans la concaténation de String, permet de gérer la concaténation et d'enregistrer les caractères dans le tas. Nous verrons son fonctionnement dans la troisième partie. La chaîne de caractère est terminée par la valeur NULL, permettant ainsi de signaler la fin de la concaténation.

Opérations avec variables

La gestion des variables se fait à l'aide de booléens et de la récupération du type de la variable dans la TDS. On traite les opérations selon les types récupérés.

Gestion des priorités

Toutes les priorités sont gérées par le fonctionnement de la grammaire et de l'AST.

5.3.5 Structures de contrôles

Rappel sur les TDS

Chacune des TDS avait un nom, dans le cas des TDS pour les `then`, `else` et `while`, les noms étaient de la forme : *nom de structure* + *numéro de if ou while*. Ce nom est très utile pour les récupérations de TDS et d'étiquette en code assembleur.

Étiquettes des structures *if*

Pour les structures `if`, on récupère le numéro de `if` que l'on utilise pour les étiquettes. On trouve 4 étiquettes :

- `if` + nb_{if} : c'est le lancement de la structure
- `then` + nb_{if} : c'est le lancement de la partie `then`
- `else` + nb_{if} : c'est le lancement de la partie `else`
- `fIf` + nb_{if} : c'est la fin de la structure

Étiquettes de boucles *while*

Pour les boucles `while`, on récupère le numéro de `while` que l'on utilise pour les étiquettes. On trouve 2 étiquettes :

- `while` + nb_{while} : c'est le lancement de la structure
- `fWhile` + nb_{while} : c'est la fin de la structure

Gestion des comparaisons

Le problème principal auquel il faut réfléchir est de pouvoir sauter la boucle while dès la première comparaison. Pour cela, on doit sauter à la fin de la boucle si la condition est fausse : on travaille donc sur les conditions inverses. Par exemple, si on croise un "<", on utilisera le suffixe **GE** pour "greater or equal".

En sachant cela, on calcule la partie la gauche de la comparaison dans R0 et la partie droite dans R1. Puis on utilise la commande **CMP R0, R1** qui effectue le calcul R0-R1 et lève les flags adaptés pour une comparaison à 0. On utilise ensuite un saut court sur cette condition inverse pour aller sur l'étiquette du **then** en cas de if ou sur l'étiquette du **fWhile** pour un while.

Enfin, on ajoute un saut court obligatoire (**BMP**) à la fin de la partie then vers l'étiquette **fIf** en cas de if pour finir la structure et vers l'étiquette **while** à la fin de la zone do dans le cas d'un while pour boucler.

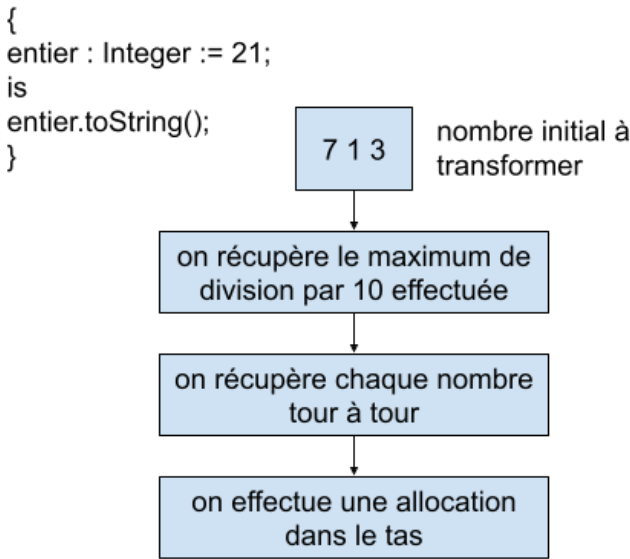
5.4 Implémentation du niveau 2 : Gestion des méthodes

5.4.1 Méthodes *print*, *println* et *toString*

Les méthodes `print()` et `println()` utilisent des trappes sur les numéros d'exception fournis. Pour ces fonctions d'écriture, on utilise l'exception `WRITE_EXC` et on affiche le contenu pointé par le registre R0. L'unique différence entre les deux fonctions est le retour à la ligne dans le `println()`. Pour convertir des entier en String, il est nécessaire d'allouer de l'espace aux Strings dans le tas, le string correspondant est la traduction en hexadécimal de l'entier. Les fonctions permettant de réaliser ces instructions sont expliquées dans la partie sur la déclaration des méthodes et sur l'allocation dans le tas.

La méthode `toString` a été réfléchié mais n'a pas pu être complètement mise en place. Transformer un entier écrit "en dur" dans l'AST est possible désormais via la même méthode que celle permettant d'enregistrer en mémoire les Strings écrit "en dur". Cependant lorsque l'entier est enregistré dans une variable, il n'est pour l'instant pas possible de le transformer en Strings.

Cependant nous avons réfléchi au potentiel fonctionnement de cette méthode. L'idée est qu'elle fonctionnera de la façon suivante. On sépare chaque chiffre du nombre complet en effectuant une boucle qui divise par 10 jusqu'à ce que le résultat de la division soit plus petit que 1 et on retient le maximum de division effectuée. On récupère ainsi tour à tour les chiffres et on les place dans le tas sous leur forme hexadécimale. On obtient donc le schéma suivant :



5.4.2 Déclaration des méthodes

Les méthodes déclarées sont identifiées par une étiquette à leur nom et générées au dessus du main. Lors de la déclaration de méthode, on fait appel à une fonction de préparation d'environnement, permettant de réserver un espace mémoire dans la pile correspondant aux variables locales de l'environnement. C'est également lors de l'appel à cette fonction que l'on fait pointer BP sur l'adresse de retour, et que l'on empile le chaînage statique. Suite à l'appel du parcours sur les instructions effectuées dans le bloc de notre méthode, on appelle la fonction `sortieEnvironnement()` qui se termine par l'instruction "RTS" pour retourner au programme principal.

Gestion des paramètres

La gestion des paramètres a lieu directement lors de l'appel de la méthode. Lors de la déclaration de méthode, on boucle sur la liste des paramètres dans le sens inverse afin de pouvoir gérer la présence de nombre négatifs. Lorsqu'on rencontre un fils "-", on le signale avec un booléen pour ne pas traiter ce noeud et on utilise la fonction `getParametreAssembleur()`, récupérant en registre sa valeur si le paramètre est un entier, ou son adresse si c'est un String ou un objet. En cas d'entier négatif, on inverse la valeur dans le registre et on saute le noeud -. On enregistre ensuite ce paramètre dans la pile. Cette gestion inverse des noeuds permet de commencer à empiler le dernier paramètre.

Gestion des valeurs de retour

Lors de la création de notre TDS de méthode, nous ajoutons une variable `result` lorsque notre méthode avait un type de retour. Afin de générer le code, on consulte la TDS de la méthode déclarée pour regarder si cette variable y est définie. Lorsqu'elle existe, on récupère son déplacement dans la méthode et on récupère sa valeur dans un registre.

5.4.3 Appels de méthode : gestion des messages

Lors de l'appel d'une méthode sur un objet, le parcours gère l'ouverture de l'environnement appelant. S'il y en a, les paramètres de la méthode sont empilés avec leur valeur. On enregistre ensuite le chaînage statique dans un registre et on réalise un saut long vers l'étiquette de la méthode appelée. Lorsque les instructions de la méthode sont exécutées, on revient au programme principal et on nettoie l'environnement appelant dans la pile.

5.5 Implémentation du niveau 3 : gestion des objets

5.5.1 Allocation dans le tas

Cas des Strings

Les cases du tas pouvant contenir deux caractères, et la concaténation ayant lieu entre deux chaînes de valeur variable, on commence par vérifier si les chaînes de caractères sont de longueur paire ou impaire. Si la longueur est impaire, le dernier caractère de la chaîne sera enregistré seul dans le tas. Si cela concerne la première chaîne, il faut sauvegarder la valeur à rajouter au début de la première lettre de la seconde chaîne.

Cette concaténation/allocation est fonctionnelle lorsque les strings ne sont pas préalablement enregistrer dans des variables et qu'ils sont lisibles directement dans les noeuds de l'AST.

Cas de l'instanciation

Les instanciations sont stockées dans le tas (à partir de l'adresse HP). Lorsque l'on instancie un objet, on enregistre l'adresse du tas. On stocke le descripteur du type dans la première case de l'objet. Enfin on augmente HP de la taille de l'objet pour enregistrer le fait que la pile avance.

5.5.2 Les constructeurs

Le constructeur fait en premier lieu une instanciation dans le tas avant d'effectuer le code compris à l'intérieur de la méthode constructeur.

5.5.3 Les descripteurs de classe

Pour chaque classe, on enregistre en TDS un déplacement, calculé en fonction des méthodes des classes précédentes. Ce déplacement sert à déterminer l'adresse du descripteur grâce à celle du début de la zone des descripteurs.

Lors de la création du descripteur, on stocke la taille d'un objet de ce type dans la première case et le nombre de méthode dans la seconde case du descripteur.

5.6 Implémentation du niveau 4

Ce niveau n'a pas encore été implémenté mais certains éléments sont déjà prêts pour avancer et nous avons déjà quelques idées d'implémentation pour faire certains de ces éléments.

5.6.1 Héritage

Pour l'héritage, il faudrait reprendre légèrement la TDS pour vérifier que les méthodes et attributs connaissent bien les mêmes déplacements dans la classe héritante par rapport à la classe héritée.

La seconde étape serait juste de bien gérer les adresses de méthodes dans le cadre des méthodes :

- enregistrement des adresses de méthodes : Il faudrait enregistrer le PC à l'initialisation de la méthode dans la case adaptée du descripteur de classe

- descripteur de classe héritante : on copie dans un premier temps toutes les adresses de méthodes communes, puis on change les adresses des méthodes re-définies (cas du **override**), enfin on ajoute les nouvelles méthodes

L'appel de méthode se ferait à partir de ce moment-là grâce au descripteur : on récupère le descripteur de classe dans la première case de l'instanciation, puis on récupère l'adresse de sous-programme dans la case adaptée à la méthode dans le descripteur. On peut maintenant lancer la méthode grâce à cette adresse.

5.6.2 Polymorphisme

En cas de cast d'un élément dans une classe parente, le principe est de remplacer l'adresse de descripteur connue par celle du descripteur de la classe parente.

On récupère bien les bonnes adresses de méthodes dans le cas de ce nouveau type dynamique.

Chapitre 6

Bilan du projet

6.1 Rapport d'étonnement 2 - Février 2021

Les rapports d'étonnement des membres ont été rédigés au cours des semaines 8 et 9, après la soutenance au sujet de la TDS et des tests sémantiques.

6.1.1 Marie-Astrid CHANTELOUP

TDS et Tests sémantiques	- La TDS est claire et très lisible. - La plupart des tests montrent les éléments importants à débogger lors de l'écriture d'un code Blood
Situation de travail	- Le lancement du travail sur la TDS a été très compliqué avec beaucoup de moment où l'équipe tournait en rond. - Ceci étant dit, une fois lancé, le travail a été très efficace et a montré l'implication de l'équipe.
Améliorations possibles	- Mieux s'informer sur le fonctionnement des transmissions d'éléments d'un langage à un autre.

6.1.2 Lucille DELAPORTE

TDS et Tests sémantiques	- TDS claire et détaillée permettant un débogage rapide et efficace du parcours et des erreur sémantique. - Implication de tous et organisation précise pour obtenir un nombre de contrôle sémantique suffisant et une liste presque exhaustive.
Situation de travail	- Démarrage difficile pour l'écriture de la TDS. - Réécriture de la grammaire qui nous a fait prendre un peu de retard sur les autres groupes pour commencer la TDS.
Améliorations possibles	- Oser demander plus rapidement de l'aide, ou de la documentation, pour ne pas être bloqué.

6.1.3 Bleunwenn GRAINDORGE

TDS et Tests sémantiques	- Bonne gestion de la TDS rendant les analyses sémantiques plus simples et efficaces - Bonne implication des membres permettant d'obtenir beaucoup de tests sémantiques
Situation de travail	- Prise de retard au démarrage car besoin de réécrire la grammaire - Démarrage difficile car peu d'idée sur comment commencer, récupérer l'AST... - Bonne répartition du travail selon les compétences et disponibilités de chacun
Améliorations possibles	- Avoir plus de documentation sur le passage de Antlr à Java

6.1.4 Niels TILCH

TDS et Tests sémantiques	- La création des tests sémantiques et de la TDS s'est très bien déroulée. - TDS et contrôles sémantiques simples et efficace à régler en cas de problème.
Situation de travail	- Des difficultés pour démarrer la TDS. - Une bonne répartition pour la création d'un maximum de contrôle sémantique.
Améliorations possibles	- Poser des questions aux encadrants en cas de difficultés majeures.

6.2 Bilan de la deuxième partie du projet

Travail attendu	Travail réalisé
- Mise en place d’une structure de TDS - Transformation de l’AST en TDS - Mise en place de tests sémantiques	- Mise en place d’une structure de TDS - Transformation de l’AST en TDS - Mise en place de tests sémantiques (environ 40 au total)
- Génération de code assembleur : <ul style="list-style-type: none">— Niveau 1 :<ul style="list-style-type: none">— déclarations de variables— affectations simples et affectations d’expressions arithmétiques— expressions arithmétiques— les structures de contrôles— Niveau 2 :<ul style="list-style-type: none">— le print d’un entier, d’une chaîne de caractères— les méthodes statiques— méthodes avec/sans valeur de retour— méthodes avec/sans paramètres— Niveau 3 :<ul style="list-style-type: none">— les méthodes d’instance, attributs et paramètres de type objet— les constructeurs— l’allocation dans le tas— Niveau 4 :<ul style="list-style-type: none">— héritage,— polymorphisme— liaison dynamique— éléments supplémentaire	- Génération de code assembleur : <ul style="list-style-type: none">— Niveau 1 :<ul style="list-style-type: none">— déclarations de variables— affectations simples et affectations d’expressions arithmétiques— expressions arithmétiques— les structures de contrôles (if while)— Niveau 2 :<ul style="list-style-type: none">— le print d’un entier, d’une chaîne de caractères— les méthodes statiques (gérées comme des méthodes usuelles)— méthodes avec/sans valeur de retour— méthodes avec/sans paramètres— Niveau 3 :<ul style="list-style-type: none">— les méthodes d’instance, attributs et paramètres de type objet— les constructeurs— les descripteurs (prêts mais non utilisés)— l’allocation dans le tas
- Mise en place d’éléments de gestion de projet	- Mise en place d’éléments de gestion de projet

Justification de la différence :

- Difficultés de communication qui ont mené à des retards sur le premier niveau.
- Retards communicatifs sur les niveaux suivants.

	Points positifs	Points négatifs	Expérience
Gestion de projet	- Comptes-rendus complets et bien détaillés - Équipe de projet agréable et complémentaire	- Problèmes de communication avec un membre du groupe - Difficulté de respect des deadlines	- Bonne communication essentielle pour le bon avancement d’un projet - Bonne rédaction des comptes-rendus de réunion permet d’améliorer l’efficacité du groupe
Écriture du code	- Deux membre du groupe à l’aise avec le code - Bonne implémentation de la TDS, des analyses sémantiques et de la génération de code	- Manque de commentaires sur le code écrit par certains membres du groupe - Beaucoup de recoupe-ments au niveau du code	- Importance des commentaires dans un travail de groupe

6.3 Bilan du projet par membre

6.3.1 Marie-Astrid CHANTELOUP

Points positifs	- Très bonne communication avec 2 membres de l'équipe - Code généralement clair et bien commenté
Difficultés rencontrées	- Très mauvaise communication avec le dernier membre de l'équipe - Légères difficultés sur microPiupk
Expérience personnelle	- Première expérience de chef de projet - Meilleure compréhension du fonctionnement d'un compilateur, notamment dans le cadre des LOO
Axes d'amélioration	- Prendre mieux en compte les personnalités de chacun en tant que chef de projet - Se renseigner sur les outils avant de les utiliser

6.3.2 Lucille DELAPORTE

Points positifs	- Bonne coopération en général entre les membres - Partage des tâches fluide - Réunion concise et précise, ToDo clairement écrite dans les comptes-rendus
Difficultés rencontrées	- Manque/problèmes de communication avec certains membres - Charge de travail exacerbée notamment avec les nombreux autres projets qui nous ont été demandé en simultané - Maladie sur les derniers jours de travail possibles
Expérience personnelle	- Expérience plus précise et plus claire du fonctionnement de la mémoire ainsi que des méthodes de compilation - Amélioration des compétences de développement en équipe
Axes d'amélioration	- Approfondir l'approche méthodique des objectifs pour limiter les manquements et les problèmes que l'on peut rencontrer lors du code

6.3.3 Bleunwenn GRAINDORGE

Points positifs	- Présence de trois élèves de IL dans le groupe, qui ont pu m'expliquer des subtilités travaillées pendant leurs cours de Traduction des Langages - Groupe efficace qui a su fournir un travail régulier
Difficultés rencontrées	- Problème de communication avec un membre du groupe - Charge de travail importante sur la fin d'année pouvant retarder l'avancement du projet
Expérience personnelle	- Meilleure compréhension du fonctionnement de la mémoire - Meilleure compréhension du fonctionnement du langage assembleur
Axes d'amélioration	- Commenter plus efficacement le code écrit

6.3.4 Niels TILCH

Points positifs	- Gestion dynamique des tâches à faire qui a permis une avancée constante dans le programme de la génération de code.
Difficultés rencontrées	- Se représenter tout ce que l'utilisateur peut écrire sur son programme - Demander de l'aide pour chercher de nouvelles solutions.
Expérience personnelle	- Meilleure compréhension de la dynamique d'un groupe de projet. - Recherche plus profonde de problèmes possibles dans le programme.
Axes d'amélioration	- Ne pas hésiter à parler clairement des difficultés rencontrées avec les autres membres du groupe de projet - Éviter de rester trop longtemps sur une partie du projet : se diversifier.

6.4 Travail réalisé

Dans cette table est consigné l'estimation du temps en heures passé sur chacune des tâches du projet.

Étapes	Marie-Astrid	Lucille	Bleunwenn	Niels
Table des Symboles	7	8	6	3
Préparation de lecture d'un fichier Blood	-	-	4	-
Ajout d'éléments	4	3	-	3
Ajout de TDS	3	3	2	-
Affichage de TDS	-	2	-	-
Analyse Sémantique	9	19	17	15
Analyse des classes	1	6	2	1
Analyse des paramètres, attributs et variables	2	2	2	11
Analyse des constructeurs	-	4	2	-
Analyse des méthodes	1	4	5	1
Analyse des blocs	3	3	2	2
Analyse des opérations	2	-	4	-
Génération de code - Niveau 1	9	15	5	14
Création parcours génération de code	-	-	2	-
Initialisation des fichiers de codes	-	0,5	-	-
Création d'environnement	1	0,5	3	1
Obtention des variables et paramètres	1	1	-	9
Affectation de valeurs	1	1	-	4
Expressions arithmétiques	-	12	-	-
Structures de contrôle	6	-	-	-
Génération de code - Niveau 2	2	15	8	8
Gestion des strings (création, concaténation, print)	-	11	-	-
Appels de méthodes	1	3	4	-
Récupération des paramètres	1	1	1	6
Valeur de retour	-	-	3	2
Génération de code - Niveau 3	10	3	1	1
Descripteurs de classe	3	2	-	1
Allocation de place dans le tas	3	1	-	-
Gestion des attributs statiques (non terminé)	2	-	-	-
Constructeur	2	-	1	-
Préparation du fichier de test de soutenance	11	2	5	9
Soutenance TDS et tests sémantique	4	1	4	3
Soutenance finale (génération de code)	7	1	1	6
Rédaction du rapport	15	5	21	10
Gestion de projet - Général	5	-	-	3
Gestion de projet - CR de réunion	3	-	5	-
TDS	1	1	4	2
Contrôle Sémantiques	1	2	4	2
Génération de code	3	1	6	1
Bilan	2	1	2	2
Total	63	67	63	60