Rapport de fin de projet

**Compilation**

Equipe C#

Equipe projet :

Florent Catiau-Tristant (Chef de projet)

Dylan Béhêtre (Responsable des tests)

Yoann Boyère

Alexis Brault

Mehidine Chupeau

Quentin Olivier

**Sommaire**

Table des matières

[Introduction 4](#_Toc472543589)

[I. Description technique 4](#_Toc472543590)

[1. Schémas d’exécution 4](#_Toc472543591)

[a. Passage de paramètres et appels de fonctions 4](#_Toc472543592)

[b. Affectations & Expressions 5](#_Toc472543593)

[c. Expressions conditionnelles 5](#_Toc472543594)

[d. Le prélude 5](#_Toc472543595)

[e. Le lanceur (Postlude) 6](#_Toc472543596)

[2. Validation du projet 6](#_Toc472543597)

[a. Validation du Pretty Printer 7](#_Toc472543598)

[b. Validation du code 3 adresses 7](#_Toc472543599)

[c. Tests de charges 7](#_Toc472543600)

[II. Architecture logicielle 7](#_Toc472543601)

[1. Les fichiers utiles au projet 7](#_Toc472543602)

[2. Organisation des packages 8](#_Toc472543603)

[a. Xtext 8](#_Toc472543604)

[b. Libwh avec BinTree 8](#_Toc472543605)

[c. Scripts de lancement 8](#_Toc472543606)

[d. Comment est transférée la table des symboles 8](#_Toc472543607)

[e. Gestion des code 3@ 8](#_Toc472543608)

[3. Tests 9](#_Toc472543609)

[III. Bilan de gestion de projet 9](#_Toc472543610)

[1. Étape du développement 9](#_Toc472543611)

[a. Nos outils 9](#_Toc472543612)

[b. Nos forces et nos faiblesses 9](#_Toc472543613)

[2. Rapport d’activité individuels 10](#_Toc472543614)

[3. Annexes 11](#_Toc472543621)

[g. Annexe 1 : schéma de l’arbre des expressions 11](#_Toc472543622)

# Introduction

Ce rapport contient les informations techniques et relatives à la gestion du projet de compilation. Ce projet de compilation entre dans le cadre du module de Compilation du semestre 7 de la formation d’ingénieurs de l’ESIR en spécialité Technologies de l’Information, option Systèmes d’Information.

### Description technique

Notre équipe a opté, lors du démarrage du projet, pour un compilateur vers le langage C#. Nous avons fait ce choix pour des questions d’affinités avec le langage et de découverte. Bien que celui-ci possède de fortes ressemblances avec le langage Java, il dispose de quelques particularités que nous souhaitions explorer.

#### Schémas d’exécution

Notre implémentation est basée sur un modèle général de traduction de code. Typiquement, le passage des paramètres pour chaque appel de fonction, quelle qu'elle soit, se traduit par un couple de files. Nous utilisons très peu les fonctionnalités présentes dans C#. Cela dit, le fonctionnement du langage ne permet pas vraiment une traduction simplifiée, en se passant de la méthode générale utilisée.

Nous détaillerons dans la suite le format de traduction pour chaque partie importante du langage WHILE.

##### Passage de paramètres et appels de fonctions

Chaque appel à une fonction (locale ou d’une librairie) induit de passer des paramètres. Afin de gérer le passage de paramètres multiples d’une manière commune, nous utilisons deux files : Queue<BinTree> inParams et Queue<BinTree> outParams. Ainsi, nous n’avons pas à gérer le nombre de paramètres : dès qu’il y a un paramètre, on l’ajoute à la file avec la méthode Enqueue(). L’opération inverse permet de récupérer ces paramètres dans leur ordre d’entrée. Ce système nous offre une gestion très simple des retours multiples des fonctions. Chaque variable à retourner est ajoutée dans la file outParams dans le corps de la fonction appelée.

|  |  |
| --- | --- |
| WHILE | C# |
| [...]  (func A B C)  [...] | [...]    inParams.Enqueue(A)            .Enqueue(B)            .Enqueue(C);    func(inParams, outParams);  [...]  func (Queue<BinTree> inputs, Queue<BinTree> outputs){    [...]    outPuts.Enqueue(ret);  } |

##### Affectations & Expressions

Lors d’une affectation (potentiellement multiple), il faut avant tout évaluer les expressions de droite avant de les affecter aux variables de gauche. On utilise alors des variables temporaires pour évaluer la valeur de ces expressions étape par étape, chacune correspondant à une variable nommée de la forme Y0.

|  |  |
| --- | --- |
| **Schéma d’exécution** | |
| WHILE | C# |
| [...]  A,B:=(func S T), (hd X)  [...] | [...]  inParams.enqueue(X);  Head(inParams, outParams);  Y0 = outParams.Dequeue();  inParams.Enqueue(S)          .Enqueue(T);  func(inParams, outParams);  Y1 = outParams.Dequeue();  A = Y0;  B = Y1;  [...] |

##### Expressions conditionnelles

Que ça soit dans une boucle ou un branchement conditionnel, les expressions servant de condition sont évaluées de la même manière. On récupère l’arbre binaire résultant de l’expression et on vérifie s’il est égal à “nil” ou non grâce aux méthodes de la librairie BinTree détaillée au point [TODO]

|  |  |
| --- | --- |
| **Schéma d’exécution** | |
| WHILE | C# |
| [...]  while ((tl A) or (hd B)) do    [...]  od  [...] | [...]  Y0 = BinTree.tail(A);  Y1 = BinTree.head(B);  Y2 = BinTree.evaluate(“OR”, Y0, Y1);  while(Y2.isTrue()){    [...]    Y0 = BinTree.tail(A);    Y1 = BinTree.head(B);    Y2 = BinTree.evaluate(“OR”, Y0, Y1);  }  [...] |

##### Le prélude

Afin de générer un programme C# compilable, il faut bien évidemment y ajouter des prérequis. Ce sont les imports et les identificateurs de classe. Ce prélude est constant, peu importe le programme while traduit.

|  |  |
| --- | --- |
| **Schéma d’exécution** | |
| WHILE | C# |
| [...] | using System;  using System.Collections.Generic;  Namespace BinTreeProject{    class Program {       [...] //Fonctions et postlude    }  } |

##### Le lanceur (Postlude)

En While, la dernière fonction du fichier constitue le point d’entrée du programme. Lors de la traduction, nous générons un lanceur qui exécute cette fonction particulière en premier, avec éventuellement des paramètres. Ce lanceur est le main() C# du programme généré. Contrairement au prélude, il est généré dynamiquement selon le programme While à traduire. En effet, pour des soucis de compréhension, nous conservons les noms des fonctions lors de leur traduction. Donc pour pouvoir exécuter la fonction principale, il faut mettre à jour le lanceur avec le nom du point d’entrée et le nombre de paramètres passé lors de l’exécution du programme généré.

|  |  |
| --- | --- |
| **Schéma d’exécution** | |
| WHILE | C# |
| [...]  function run:  read X  %  %  write X | public void run(Queue<BinTree> input, Queue<BinTree> output){    [...]  }  public void main(argc, args[]){    //Conversion des arguments en BinTree    X0 = convertStrToBinTree(args[0]);    //Même action pour les éventuels autres paramètres    run(inParams,outParams);    outParams.dequeue();    //Récupération du résultat de la fonction  } |

#### Validation du projet

Afin de valider les différentes fonctionnalités composant notre projet, nous avons mis en place de batteries de tests unitaires.

Chacun de ces ensembles de tests correspond à une partie spécifique du développement de notre projet (Pretty printer, code 3 adresses, table des symboles, ..).

Elles nous ont permis de vérifier dans un premier temps le bon fonctionnement de nos développements, en validant le résultat produit. Puis dans un deuxième temps, de vérifier la robustesse et la conformité de notre code lors de modifications.

##### Validation du Pretty Printer

Dans un premier temps, et ce, en parallèle de la réalisation du Pretty Printer, nous avons mis en place 40 tests unitaires permettant de vérifier la validité du traitement réalisé.

Pour ce faire, nous avons établi deux types de tests. Les premiers consistent à Pretty Printer un fichier d’entrée (appelé original), dont le contenu concerne un fragment spécifique de la grammaire et dont la mise en page est chaotique. Le résultat produit (appelé résultat) est alors validé en comparant son contenu avec un oracle (un fichier correspondant au résultat attendu).

Les deuxièmes consistent à vérifier, via la récupération d’une exception, qu’un fichier original ne correspondant pas à la grammaire WHILE lève une exception.

##### Validation du code 3 adresses

[TODO]

##### Tests de charges

[TODO]

### Architecture logicielle

#### Les fichiers utiles au projet

Les fichiers indispensables et fixes :

* **whc.exe** : Script de lancement du compilateur avec différentes options
* **whpp.exe** : script de lancement du pretty printer avec différentes options
* whc.jar : jar exécutable lançant le compilateur
* whpp.jar : jar exécutable lançant le pretty printer
* whc\_lib : répertoire contenant les librairies (.jar) utilisées par whc.jar
* whpp\_lib : répertoire contenant les librairies (.jar) utilisées par whpp.jar

*Tous ces fichiers doivent être placés dans le même répertoire sous-peine de ne plus fonctionner (ou impliquant une modification des .jar et des sources des scripts.)*

* BinTree.cs : Source C#. Librairie libwh décrite au [point b.](#_Libwh_avec_BinTree)

*Ce fichier peut être déplacé mais demande une légère modification du source du script whc.cpp afin de recompiler l’exécutable final.*

Certains fichiers sont auto-générés lors de l’exécution du compilateur :

* Le fichier de sortie .cs (source C#) indiqué en option au compilateur (out.cs par défaut)
* Le fichier exécutable .exe indiqué en option au compilateur (out.exe par défaut)
* Le fichier source While indiqué en option au pretty printer (sth.whpp par défaut)

#### Organisation des packages

##### Xtext

La première chose que l’on a dû créer pour ce projet est la grammaire du langage While. Le langage en lui-même n’est pas compliqué mais on a dû s’y reprendre à plusieurs reprises pour obtenir le résultat voulu. Cette grammaire nous permet de récupérer les différentes parties du programme (parser) et de le décomposer comme nous le souhaitons pour que l’on puisse générer le code 3 adresses correspondant à chaque morceau du programme. Notre grammaire contient plusieurs règles permettant de récupérer les différentes parties du code.

##### Libwh avec BinTree

Afin de permettre la traduction d’un programme While en C#, nous avons dû créer une librairie BinTree.cs. Cette librairie contient le type de variable qui sera utilisé dans le programme C# final. Elle contient toutes les méthodes utiles à l’utilisation de ces variables telles que cons, list, head ou tail. Nous pouvons aussi évaluer ces BinTree via la fonction evaluate() avec trois paramètres différents : EQ (égal), AND et OR. Nous nous basons sur cette librairie pour la traduction des instructions While, elle est donc indispensable au bon fonctionnement du code généré en C#.

##### Scripts de lancement

Nous avons créé plusieurs scripts de lancement pour nos différents exécutable que ce soit pour le pretty printer ou pour le compilateur. Ces scripts ont été fait à base de C++ afin de pouvoir les compiler pour les différents OS. Notre compilateur étant fait en Java, il est donc multi-plateforme donc il y avait un fort intérêt à faire un script de lancement lui-même multi-plateforme.

Les scripts de lancement permettent d'exécuter les différents programmes avec des options personnalisant l’exécution. Chaque script de lancement dispose d’un manuel d’utilisation, une présentation des options et la présentation des auteurs.

##### Comment est transférée la table des symboles

La table des symboles est une liste de fonctions. Chaque fonction (identifiée par son nom) dispose d’un objet propre “DefFun”, qui comporte toutes les informations relatives aux symboles présents dans cette fonction. Ainsi, il est aisé de récupérer toutes ces données lors de la génération du code dès lors qu’on dispose du nom de la fonction à traiter.

##### Gestion des code 3@

La production du code intermédiaire est technique et assez complexe à mettre en place. La partie la plus difficile a été la gestion des expressions. Nous avons décidé d’utiliser des arbres pour gérer l’ensemble des expressions. Voir annexe 1 pour une explication schématique du fonctionnement interne de l’arbre des expressions en montrant sur un exemple.

Notre arbre empile les différents symboles et variable, puis une fois l’expression terminée, simplifie l’arbre en sous arbre et génère du code 3 adresses. Pour les fonctions acceptant des listes de fonctions comme “cons” nous avons décidé de faire des “push” afin de recréer cette liste avec seulement un code 3 adresses. Le système inverse avec des pop en plus pour les symboles “fonctions” définis dans le programme while à compiler.

Les quadruplets générés sont mis dans une liste d’étiquette, elle-même gérée avec une pile dynamique pour gérer les imbrications.

Toutes ces informations sont accessibles depuis une Map qui associe les étiquettes générées avec les quadruplets qu’elles contiennent. Lors de la génération du code, nous parcourons simplement la liste de ces codes 3 adresses et les traduisons selon leur nature.

#### Tests

Pour les tests, nous avons utilisé JUnit. JUnit est un framework de test unitaire compatible avec Java. Celui-ci nous a permis d’écrire des scripts de test propre à chaque composante de notre système. Ainsi, vous pouvez trouver dans le répertoire “testUnitaire” du projet “esir.compilation”,  les packages “prettyPrintTest” et  “traductionTest” correspondant respectivement à la partie “PrettyPrint” du projet et à la partie “traduction”.

Dans le package “prettyPrintTest”, on trouve le script de test correspondant nommé “PrettyPrintTest.java” ainsi que 3 dossiers, un premier contenant les fichiers à traiter (pretty-printé), un deuxième contant les oracles (résultats attendus) et un troisième contenant les fichiers après leur traitement.

Le package “traductionTest” est similaire dans sa structure. Nous avons pour le code 3 adresses, la table des symboles et la traduction finale des scripts spécifiques, avec pour chacun, un dossier contenant les fichiers en entrés, qui lui sont rattachés.

// A COMPLETER

### Bilan de gestion de projet

#### Étape du développement

##### Nos outils

Nous nous sommes organisés de la manière suivante :

* Utilisation de Git via le logiciel Source Tree. Il était indispensable pour nous d’utiliser Git. En effet, cet outil nous permet de pouvoir travailler à plusieurs sur le projet sans risquer quoi que ce soit (perte de code). Nous pouvons facilement savoir qui a fait quoi récemment ou bien décidé d’effectuer une branche à part pour un travail spécifique, etc.
* Trello est un très bon outil de gestion de projet. Il permet de savoir où nous sommes rendus dans le projet, ce qui est à faire, ce qui est en train d’être fait et par qui ainsi que ce qui a été fait. Nous pouvons avoir facilement un aperçu de l’avancé de notre travail sur le projet.

Ces outils nous ont donc permis de pouvoir gérer notre projet plus facilement.

##### Nos forces et nos faiblesses

Même si nous avons eu quelques difficultés quant à l’utilisation de SourceTree, il nous a quand même permis (ainsi que Trello) de bien organiser notre projet et cela a plutôt bien marché. D’un autre côté, nous n’avons pas pris assez de temps pour faire des réunions sur l’explication de ce que chacun a fait. Cela nous aurait permis de plus facilement comprendre les parties du projet ou certains d’entre nous allaient aider d’autres.

#### Rapport d’activité individuels

##### Rapport d’activité de Florent Catiau-Tristant : chef de projet

##### Rapport d’activité de Dylan Béhêtre : responsable des tests

Lors du développement du projet, mon principal rôle à été de réaliser les tests unitaires.

J’ai ainsi défini l’architecture du test. La technologie comme la construction de ceux-ci.

J’ai également réalisé la majorité des tests unitaires. C’est à dire les tests du PrettyPrinter. Avec l’aide de Quentin pour la réalisation des oracles. Les tests dédiés au code 3 adresses dans leur ensemble. Les tests dédiés à la table des symboles. Ainsi que les tests dédiés à la transformation d’un code While en un code C#.

J’ai également participé aux réflexions en amont sur le schéma de traduction et le code 3 adresses. J’ai essayé de suivre les avancés des membres de mon groupe afin d’en comprendre les enjeux comme l’objectif. Cela m’a également permis de réaliser les tests appropriés.

De plus, j’ai participé à la rédaction du rapport de fin de projet. J’ai réalisé la mise en forme et le cadrage. J’ai participé à la réalisation de l’introduction. J’ai parlé de la partie test au sein de la partie architecture logicielle.

##### Rapport d’activité de Yoann Boyere

Mon activité a commencé lors du sprint1 en écrivant la grammaire while dans le projet xText. J’ai ensuite aidé le reste du groupe pour effectuer le pretty printer.

Ensuite, au début du sprint2, je me suis attelé à rédiger avec Quentin le schéma de traduction. Celui-ci a continué à être modifié lors de ce sprint ainsi que le 3ème. J’ai de mon côté aidé Mehidine à continuer la partie “pretty printer”.

Assez vite, me rendant compte que les deux autres importantes parties (table des symboles et code 3 adresses) étaient bien avancées et maîtrisées par Florent et Alexis, je suis très vite passé sur le dernier sprint en commençant la librairie en c# : BinTree.cs. Cette classe est obligatoire afin de pouvoir créer les variables (des arbres) ainsi que les évaluer (cons, list, and, or, etc).

Une autre partie importante du sprint était la traduction du code 3 adresses en C#. Alexis m’a donné les bases de cette traduction afin de mieux appréhender son travail sur le code 3 adresse. A partir de ça, j’ai donc travaillé sur la traduction vers le code C# (création des fonctions, utilisation des reads et writes, le main, passages des arguments,etc). Mehidine et Alexis ont eux aussi contribué à cette traduction.

Enfin, j’ai aidé à la résolution des bugs concernant la génération du code final, faire le script de compilation du fichier final c# et à la rédaction de ce document.

##### Rapport d’activité de Alexis Brault

Ma contribution dans le projet a commencé par la production de l'exécutable whpp avec ses différents paramètres.

Je me suis ensuite greffer au code de Florent pour générer le code 3 adresses. Toute la première partie, je l’ai réalisé seul puis Méhidine est venu m’aider pour compléter les parties du code 3 adresses pour lequel je n’avais pas eu le temps d’avancer.

D’autre part, j’ai conçu le code permettant de générer le code C# à partir du code 3 adresses généré auparavant. Puis j’ai donné les clés de ce nouveau code à Yoann pour qu’il puisse mieux l’appréhender le compléter avec son avancement sur la libwh.

De plus, j’ai essayé d’aider au mieux que je pouvais Dylan pour qu’il puisse réaliser ses tests même si cet échange n’a pas été des plus simple, difficulté de ma part à expliquer ce que j’ai fait et surtout le vulgariser à la juste valeur pour qu’il puisse faire les tests nécessaires.

Et enfin, j’ai participé à la rédaction de ce livrable, à la communication dans le groupe ainsi que la mise en place des outils ainsi que la mise à jour de leur contenu.

##### Rapport d’activité de Mehidine Chupeau

J’ai commencé à prendre part dans le projet dès le sprint 1 en programmant le pretty printer avec l’aide de Yoann qui a écrit la grammaire du langage while. J’ai par la suite dû améliorer la grammaire et le pretty printer lors de ce sprint 1 car il y avait des problèmes que l’on n’a pas su voir mais que les tests de Dylan et Quentin ont révélés.

A la fin du pretty printer, Florent, Alexy et Yoann avaient déjà commencé le sprint 2, je me suis donc tourné vers Alexis qui n’avait pas encore fini le code 3 adresses pour l’aider à le terminer. J’ai donc pu programmer le code 3 adresses des expressions booléennes ainsi que le code 3 adresses des boucles for, foreach, while et if. Comme nous travaillons avec la méthode agile, à la fin de chaque fonction que je codais dans le code 3 adresses, je codais en même temps sa traduction en C# dans le fichier CS\_translator.java.

Durant toute la durée du projet, je suis resté en contact avec les tous les membres du groupe pour que je puisse leurs apporter de l’aide dans les différents bugs rencontrés mais plus particulièrement avec Dylan qui était le responsable des tests pour qu’il puisse me dire s’il existe des problèmes dans le code que j’ai écrit.

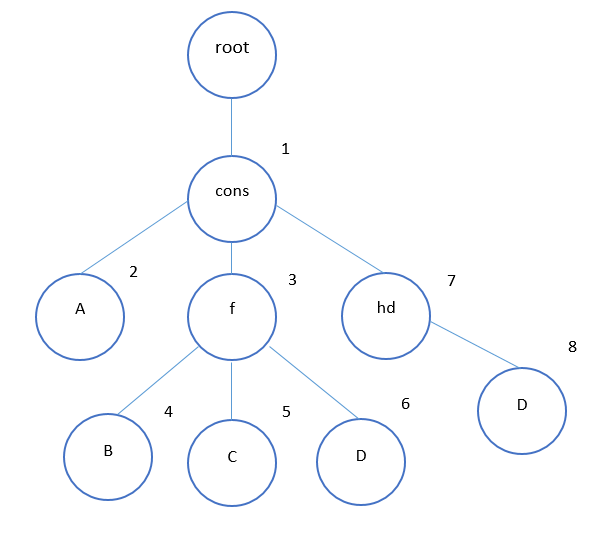
Vers la fin du sprint 3, j’ai passé du temps avec Alexis, Florent et Yoann pour que l’on puisse régler des petits détails dans le code 3 adresses et la génération de code et notamment la classe BinTree en C# pour que le compilateur soit fonctionnel le jour de la présentation.

##### Rapport d’activité de Quentin Olivier

#### Annexes

##### Annexe 1 : schéma de l’arbre des expressions

Soit l’expression : A := (cons A (f B C D) (hd D))  
, f étant une fonction a 3 entrées et 2 sorties. La numérotation correspond à l’ordre d’entrée dans l’arbre.



Voici les étapes de simplification :

(f B C D) est devenu Y0 et Y1; (hd D) est devenu Y2; soit A := (cons A Y0 Y1 Y2)

|  |  |
| --- | --- |
| ExprTree | Code 3 adresses généré : |
| https://lh3.googleusercontent.com/LsKG9vY9BNfgamX0fv7u4ntfVb_MQgzFD_dA7vRndS_Quf0_j5Brnn0pmO8632JG38vHbrj8c0cCNv5gr0cj0zFrkE_A9_U6evC8ivSjyiFX_fO2JvQIRyRh6s1aiYvpxGgAlili | <PUSH, \_, B, \_ >  <PUSH, \_, C, \_ >  <PUSH, \_, D, \_ >  <CALL f, \_, \_, \_ >  <POP, Y0, \_, \_ >  <POP, Y1, \_, \_ >  <HD, Y2, D, \_ > |
| https://lh5.googleusercontent.com/25JW7okNFg0o_S5wXukeVdVn7EFT-_NSf8NKZ9XPjZGV7eCSpVDbMfONuPImHTEdXPCXFTXudzpek36F3d4fqPzyDtSdIvEOuPyTPxZX8YH8gWqX64Gw9aev4ALs00SLc70OCJl3 | <PUSH, \_, A, \_ >  <PUSH, \_, Y0, \_ >  <PUSH, \_, Y1, \_ >  <PUSH, \_, Y2, \_ >  <CONS, Y3, \_, \_ > |