ENSSAT
LANNION

UNIVERSITÉ DE RENNES 1

Projet de Compilation
Réalisation d'un compilateur
Pour le langage nilNovi Objet

I – Organisation du projet

A. Composition de l'équipe & rôles

Chef de projet : Hugo Chataigner
Membres du projet :

- Benjamin Danjoux
- Thomas Liberge
- Pierre-Antoine Moitier
- Annaïg Peneau
- Louis-Maxime Piton

B. Répartition des tâches

Nous nous sommes répartis les tâches de la façon suivante. Nous avons initialement tous étudié le sujet, puis avons chacun codé des points de génération de la partie NNA du projet, tout en maintenant à jour un index des points déjà traités. Une fois qu'une majorité des points de génération fut implémentée, Louis-Maxime, Benjamin et Hugo ont terminé les points de génération les plus complexes à mettre en œuvre, tandis que Pierre-Antoine, Thomas et Annaïg se sont concentrés sur le codage des erreurs. Enfin, nous nous sommes tous attelés à la rédaction du rapport.

C. Calendrier prévisionnel et effectif

	Analyse du problème	**Codage des points de génération**	**Tests**	**Rédaction du rapport**	**TOTAL**
Annaïg	10:00:00	09:00:00	07:00:00	04:00:00	30:00:00
Hugo	08:00:00	20:00:00	1:00:00	1:00:00	30:00:00
Loulou	10:00:00	15:00:00	3:00:00	2:00:00	30:00:00
Pierre-Antoine	10:00:00	11:00:00	05:00:00	4:00:00	30:00:00
Benjamin	8:00:00	20:00:00	01:00:00	1:00:00	30:00:00
Thomas	10:00:00	09:00:00	06:00:00	5:00:00	30:00:00
TOTAL	56:00:00	84:00:00	23:00:00	17:00:00	180:00:00

II – Description du processus de compilation NN

A. Description générale

Le compilateur NN met en place plusieurs étapes qui vont au fur et à mesure analyser le code à compiler. La première étape consiste en l'analyse lexicale du programme fourni. Celle-ci a pour but de lire le code brut et de le découper en unités lexicales utilisables plus tard par le compilateur. La seconde étape est l'analyse syntaxique, celle-ci va utiliser les unités lexicales précédemment générées afin d'en ressortir les différents blocs de code du programme fourni. Cette analyse va permettre des appels aux Points de Génération afin de créer du code compilé. Au fur et à mesure de l'analyse syntaxique, le compilateur va stocker des informations sur toutes les entités rencontrées dans une table des identificateurs, afin de pouvoir utiliser ces informations plus tard dans l'analyse.

B. Analyseur lexical

L'analyseur lexical permet de découper le programme brut en unités lexicales. Une unité lexicale est un bloc de code qui a un sens particulier pour le programme comme par exemple un nom de variable ou un mot clé. Pour faire cela, l'analyseur va parcourir le code brut jusqu'à découvrir un bloc cohérent et va par la suite le stocker dans la table des identificateurs pour l'utiliser si besoin dans les étapes suivantes.

C. Analyseur syntaxique

L'analyseur syntaxique va utiliser les unités lexicales récupérées de l'analyseur lexical et il va donner un sens à leur organisation. Il va analyser les différents blocs de code afin de savoir comment ils sont utilisés dans le programme. Par exemple, il repère les débuts et fins de programme ou de fonctions, les blocs de programmation comme les "if", "else", ou "while" pour les assembler convenablement. Il gère les variables et leurs appels plus loin dans le programme. Pour cela, il peut créer de nouvelles instructions machine en faisant appel à des points de génération. Il stocke aussi de nombreuses informations dans la table des identificateurs comme par exemple le nom des variables et la position de leurs données dans la pile de données afin de pouvoir, plus tard, mettre en place leur appel.

D. Table des identificateurs

Elle a pour but de stocker les informations relatives aux différentes entitées rencontrées au cours de la compilation du programme. Pour cela, elle va stocker de nombreuses informations lors de la création et de l'appel des variables, ainsi que lors de l'initialisation des classes et de leurs fonctions et procédures. Toutes ces informations vont servir plus tard dans la compilation quand il sera nécessaire d'appeler des fonctions ou variables.

III – Modifications/Ajouts apportés au projet.

A. NNA

Les principales parties à ajouter à NNA étaient la gestion globale de la forme du programme et la gestion des variables et de leurs calculs. Pour la gestion globale de la structure (begin, end, …), nous avons suivi les fonctions données dans l'analyseur lexical et nous avons ajouté des fonctionnalités aux points de génération afin d'ajouter les instructions compilateur au fur et à mesure de l'analyse.

Ensuite pour la partie gestion des variables, un groupe s'est chargé de l'attribution des variables en mémoire pendant que l'autre gérait la partie analyse des expressions.

Dans un second temps une équipe a travaillé sur la mise en place de la vérification des erreurs du code donné par le développeur.

B. NNO

Dans le cas de NNO de nombreuses parties ont dû être ajoutées. Nous avons commencé avec la structure générale de classe afin de pouvoir par la suite continuer en programmant la gestion des procédures et des fonctions.

Une des particularités de la programmation objet est qu'elle propose des fonctionnalités d'héritage, les fonctions héritées de la classe mère sont accessibles dans la classe fille mais peuvent également être redéfinies. Pour permettre cela, il faut après la déclaration des fonctions et procédures, attendre la fin de la partie où celles ci sont implémentées. En temps normal on vérifie simplement que toutes les fonctions déclarées ont été implémentées, en NNO, il faut identifier les méthodes non implémentées, vérifier si celles-ci sont héritées, auquel cas il faut indiquer leur adresse d'implémentation comme étant l'adresse d'implémentation dans la classe mère. Ainsi, on ne lève une exception que si une méthode est non implémentée et qu'elle n'est pas héritée.

Nous n'avons pas complètement fini le travail sur l'héritage entre les objets ni l'ajout de fonctionnalités propres à NNO comme le mot-clé "this" ainsi que la gestion des erreurs. Cette partie n'a pas pu être totalement aboutie dans le temps imparti.

Concernant la structure classe, nous avons travaillé de la même façon que pour la structure de programme en suivant l'analyseur syntaxique. Des modifications ont dû être apportées à la gestion des variables afin de rajouter l'initialisation de classe dans une variable.
Le travail sur la gestion des procédures et des fonctions a ensuite commencé.

Pour cette partie nous avons dû rajouter de nombreux flags dans les points de génération afin de différencier les différents cas entre fonctions, constructeur et procédures. Il a aussi fallu faire attention à la gestion des différents types de variables (locale, globale, attributs) ainsi que leur type (donnée, objet).

IV – Résultats & Discussions

A. Jeu d'essai NNA

1) Description du jeu d'essai NNA

Nous avons utilisé le jeu d'essai NNA fourni, car il comprenait la majorité des erreurs à tester. Nous les avons ponctuellement modifiés pour vérifier que les erreurs retournées l'étaient dans tous les cas de figure.

2) Résultats

n° programme	Objectif du test	X	Remarques
	Programmes corrects		
correct1	Boucles imbriquées	X	
correct2	Expression relationnelle avec des booléens	X	
correct3	Evaluation de conditions complexes	X	
correct4	Affiche le nombre de valeurs paires lues	X	
	Programmes incorrects		
error1	Lecture d'un booléen	X	
error2	Ecriture d'expressions booléennes	X	
error3	Affectation : typage incorrect		
error4	Condition incorrecte dans une alternative	X	
error5	Condition incorrecte dans un *while*	X	
error6	Double déclaration de variable	X	

3) Discussions sur les résultats & critique des choix effectués

La totalité des erreurs et des programmes corrects a été gérée sans soucis.

B. Jeu d'essai NNO

1) Description du jeu d'essai NNO

Nous avons utilisé le jeu d'essai NNO fourni, car il comprenait la majorité des erreurs à tester. Nous les avons ponctuellement modifiés pour vérifier que les erreurs retournées l'étaient dans tous les cas de figure.

2) Résultats

n° programme	Objectif du test	X	Remarques
error1	Redéfinition d'une méthode-procédure par une méthode-procédure avec un profil différent, nombre de paramètres différents		pas d'erreur retournée
error2	Redéfinition d'une méthode-procédure par une méthode-procédure avec un profil différent, type de paramètres différents		pas d'erreur retournée
error3	Redéfinition d'une méthode-procédure par une méthode-procédure avec un profil différent, mode de passage différent		pas d'erreur retournée
error4	Redéfinition d'une méthode-procédure par une méthode-procédure avec un profil différent, nom de paramètres différents		Arrêt prématuré de la compilation - débordement du programme objet
error5	Redéfinition d'une méthode-fonction par une méthode-procédure		pas d'erreur retournée
error6	*this* déclaré comme identificateur de paramètre formel		Arrêt prématuré de la compilation - débordement du programme objet
error7	*this* utilisé comme identificateur d'attribut		pas d'erreur retournée
error8	*this* utilisé comme identificateur de classe		erreur retournée mais pas de description
error9	Redéfinition d'un attribut dans une même hiérarchie		pas d'erreur retournée
error10	*integer* comme identificateur d'attribut		pas d'erreur retournée
error11	*integer* comme identificateur de classe		erreur retournée mais pas de description
error12	Classe sans constructeur	X	
error13	Double déclaration de paramètre formel dans une méthode		pas d'erreur retournée
error14	Identificateur d'attribut doublement déclaré		pas d'erreur retournée
error15	Double déclaration d'un identificateur de classe		erreur retournée mais pas de description
error16	Redéfinition d'un paramètre par une variable locale d'une méthode		erreur retournée mais pas de description
error17	Redéfinition d'une opération dans la même classe		pas d'erreur retournée
error18	Conflit attribut/classe		pas d'erreur retournée
error19	Appel d'une méthode incompatible avec sa définition		pas d'erreur retournée
error20	Affectation *this* dans une méthode		erreur retournée mais pas de description
error21	Appel d'un constructeur incompatible avec sa déclaration : appelé avec une classe différente de celle ou il est défini		ne compile pas
error22	Appel d'un constructeur incompatible avec sa déclaration : appelé avec un objet (et non une classe)		ne compile pas

error23	La définition d'une méthode est différente de sa déclaration : fonction devient procédure		erreur retournée mais pas de description
error24	Type délivré par une fonction différent du type déclaré		pas d'erreur retournée
error25	Problème de *return*		Fichier n'existe pas
error26	Appel d'une méthode spécifique au type dynamique		pas d'erreur retournée
error27	Paramètre formel de mode *in out* : paramètre effectif de mode *in*		erreur retournée mais pas de description
error28	Une méthode déclarée n'est pas définie dans la classe		ne compile pas
error29	Fonction avec paramètre de mode *in out*		pas d'erreur retournée
error30	Constructeur avec paramètre de mode *in out*		ne compile pas
error31	*return* dans un constructeur		ne compile pas
error32	*return* dans la partie impérative		pas d'erreur retournée
error33	Héritage depuis une classe qui n'existe pas		erreur retournée mais pas de description
error34	Plusieurs constructeurs		erreur retournée mais pas de description
error35	Pas de constructeur	X	
error36	*return* dans une méthode-procédure		pas d'erreur retournée
error37	Accès à attribut hors classe		erreur retournée mais pas de description
error38	*this* utilisé en dehors d'une classe		ne compile pas
error39	Type délivré effectivement supérieur au type déclaré		pas d'erreur retournée
error40	Double déclaration de méthode-procédure		erreur retournée mais pas de description
error41	Affectation de variables objet incompatibles		pas d'erreur retournée
error42	Un constructeur appelé comme une procédure		pas d'erreur retournée
error43	Fonction qui modifie un paramètre formel		erreur retournée mais pas de description
error44	Pas de *return* dans le corps d'une fonction		pas d'erreur retournée
error45	Classe qui hérite d'elle même		ne compile pas

n° programme	Objectif du test	X	Remarques
correct1	Redéfinition correcte d'un identificateur d'attribut par un identificateur de constructeur	X	
correct2	Redéfinition correcte d'un identificateur d'attribut par un identificateur de variable locale	X	
correct3	Test de "error"	X	
correct4	Essai de polymorphisme et de liaison dynamique	X	
correct5	Manipulation de listes d'objets		ne compile pas
correct6	Imbrication d'appels de méthodes-fonctions		Arrêt prématuré de la compilation - débordement du programme objet
correct7	Méthodes identifiées par integer et par nil		Programme nilNovi erroné (même dans le fonctionnel)
correct8	Essai pronom *this* et passage de paramètre		Programme nilNovi erroné (même dans le fonctionnel)
correct6	Définition de méthodes dans un ordre différent de leur déclaration	X	
correct10	Évaluation d'expressions arithmétiques en NNO		ne compile pas
correct11	Expression d'objet complexe : appel d'une fonction qui délivre un objet		Programme nilNovi erroné (même dans le fonctionnel)
correct12	Appel récursif d'une méthode-procédure et pronom « *this* »		Programme nilNovi erroné (même dans le fonctionnel)

3) Discussions sur les résultats & critique des choix effectués

Comme nous n'avons pas eu le temps de finir la partie héritage et "this", une majorité des erreurs de NNO n'a pas pu être testée. De plus, nous avons remarqué que certains programmes de test, comme par exemple correct7, étaient erronés, en effet, leur compilation échoue même en utilisant la version fonctionnelle fournie avec le projet. Nous n'avons pas eu le temps d'essayer de comprendre les erreurs de ces programme test.

V – Perspectives & conclusion

A. Etat du programme

Le compilateur est actuellement capable de totalement gérer NNA et les erreurs qui lui sont liées. Pour la partie NNO une grande partie de la compilation des objets est fonctionnelle. Il est possible de créer des objets avec des procédures et des fonctions et de les appeler dans le programme principal. La partie héritage est en cours de développement mais ne fonctionne pas encore totalement. Une partie des erreurs lors de l'utilisation de NNO est gérée.

B. Possibles évolutions du programme

Cependant, il reste plusieurs points à déboguer sur la version actuelle du programme. Une partie de NNO reste à développer, principalement les fonctionnalités liées à l'utilisation de l'héritage et de "this". Il faut finir de gérer une partie des erreurs pouvant être levées avec NNO. À la suite de cela une réorganisation du code peut être nécessaire afin d'améliorer la lisibilité pour de futurs autres développeurs, et il pourra aussi être nécessaire de vérifier le code du compilateur afin de nettoyer une partie des actions inutiles afin d'accélérer son fonctionnement.

C. Difficultés majeures

La première grande difficulté que nous avons rencontré a été de comprendre ce que nous avions à réaliser lors de ce projet. En effet, nous avions vu que certains points de génération étaient déjà implémentés, et nous ne savions donc pas ce que nous avions à coder.

La deuxième difficulté a été de déterminer quelles informations sauvegarder dans la table des identificateurs et à quel moment.