CAHIER DE RECETTE

Version :	*< 1.0.3>*
Date :	*< 15/04/2015>*
Rédigé par :	*< IDRISSOU Amzath>*
Relu par :	

MISES A JOUR

Version	Date	Modifications réalisées
0.1	19/11/14	Création
0.2	12/12/14	Modification complète des procédures de tests : Ajout de procédures pour chaque exigence de réalisation de la STB.
0.3	20/12/14	Détails plus spécifiques de plusieurs procédures de tests.
1.0	27/12/14	Relecture et correction orthographique.
1.01	18/01/15	Corrections apportées suite aux retours des professeurs : - Ajout de la procédure de test pour l'EQ_1 - Explication plus détaillé de : o Introduction o Environnement de test o Responsabilités o Stratégie des tests o Gestion des anomalies
1.0.2	27/03/2015	Corrections apportées suite au nouveau **PDD** et **DAL** • Suppression de certains cas de tests • Ajout de nouveau cas de Tests • validation de certains cas de tests
1.0.3	15/04/2015	Mise à jour de la CDR suite à la nouvelle version de le STB 1. suppression du cas de test compilation appliaction en mode partagé 2. suppression cas de test compiler bibliothèque partagée 3. suppression du cas test Indiquer les chemins des dépendances entre le source de l'application et des modules (classe/interface) externes 4. suppression du cas de test définition d'attributs ou de variables de type value 5. suppression du cas de test définition méthode value 6. suppression du cas de test gestion exceptions

Introduction :

- *Fonctionnalités du logiciel (liste des cas d'utilisation)*

  - Afficher aide
  - Compiler une application en mode monolithique
  - Compiler une application en mode partagé
  - Compiler une application utilisant des bibliothèques partagées
  - Indiquer les chemins des dépendances entre sources et classes déjà compilées
  - Afficher version compilateur
  - Activer affichage en couleur

- *Liste des objets à tester (versions, lots, modules, etc.)*

  A la livraison de chaque livrable de notre compilateur un certain nombre de tests va devoir être évalué.
  Remarque : il y aura 3 livrables réalisés :
  - Compilation en mode monolithique
  - Compilation en mode partagé
  - Compilation en mode partagé avec des bibliothèques partagées

  Le lot de tests a évalué ainsi que la version dépendra de l'évolution du compilateur.

- *Contexte d'exécution des tests*

  Nous allons nous servir du logiciel SublimeText pour coder les tests.
  En ce qui concerne le matériel nécessaire seul un terminal sur l'ordinateur de l'utilisateur sera nécessaire, en plus des fichiers sources et du logiciel kawa installé.

- *Choix technologiques et dispositions particulières*

  Git, LLVM, Kawa

1. Documents applicables et de référence

Le document de spécification du client Spec1.pdf
La STB
Le site de LLVM : « **LLVM.org** »

2. Terminologie et sigles utilisés

- **Kawa :** Langage jouet qui reprend quelques fonctionnalités de java.
- **Clang :** Compilateur pour le langage c++, son interface de bas niveau se base sur des bibliothèques.
- **C++ :** Langage de programmation orienté objet bas niveau.
- **Makefile** : Fichier regroupant des instructions de compilation avec gestion de dépendances.
- **Ubuntu :** Distribution linux sur base Debian.
- **LLVM** (Low Level Virtual Machine) : Infrastructure de compilateur conçue pour l'optimisation de la compilation.
- **ELF** (Executable and Linkable Format) : Format de fichier binaire utilisé pour l'enregistrement de code compilé (objets, exécutables, bibliothèques de fonctions).
- **Makefile :** Fichier regroupant des instructions de compilation avec gestion de dépendances.
- **POO** : Programmation orientée objet.

3. Environnement de test

- *Configurations matérielles utilisées*
  Un simple ordinateur avec Kawa installé suffit pour réaliser les tests.
- *Outils de test mis en œuvre (éventuellement)*
  -
- *Jeu de données et/ou bases de données de test*
  Il n'y aura pas de jeu ou de base de données au sens stricte du terme. Mais nous allons créer des classes auto-générées avec des données aléatoires à chaque réalisation de test. Les données ne seront pas sauvegardées sauf en cas d'erreur, afin de trouver l'origine du problème

4. Responsabilités

Le testeur a l'entière responsabilité des tests. Néanmoins, il peut s'associer à un ou plusieurs techniciens Java afin que ceux-ci lui fournissent des modèles facilement testables.
Il peut, lui aussi, indiquer ce qui doit être testé.

C'est lui qui doit clôturer le codage des fonctionnalités, et seul lui peut dire si elle marche correctement dans son intégralité et permettre aux développeurs de se concentrer sur les taches suivantes.

En cas d'erreur non signalé par le testeur, mais découvert lors du lancement du projet c'est le testeur qui est responsable de ce souci. Il en va de sa responsabilité, et en cas d'erreur cela peut mettre l'équipe dans une situation délicate vis-à-vis du client.

5. Stratégie de tests

Avant toute démarche de tests, il est nécessaire de s'assurer que la partie testée n'est pas en cours de développement. Pour ce faire, un planning doit être instauré et respecté.

Pour chaque cas de test on s'arrêtera quand on aura un résultat contraire au résultat attendu ou si on obtient le résultat attendu après compilation.

Il sera nécessaire de réaliser des tests dits « unitaires » (c'est-à-dire pour chacune des fonctionnalités) ainsi que des tests « d'intégration » (c'est-à-dire qui vérifieront l'intégralité des fonctionnalités implémentées à un instant t, pour en faire un livrable).
Le dernier type de test est le test de « non régression ». Ces tests seront effectués lors d'un ajout ou de la modification d'une fonctionnalité qui peut avoir eu un impact sur d'autres fonctionnalité. Ces tests vérifieront que l'existant fonctionnel n'a pas été « détérioré ».

A chaque fois qu'un test est réalisé il faut notifier le résultat obtenu dans le tableau des procédures (qu'on trouve plus bas dans ce document).

L'objectif étant d'avoir tous les tests validé à la fin du projet. Et que ces tests touchent le plus de ligne de code possible; c'est ce qu'on appelle la couverture de code.

NB : en cas de modification d'une partie de code par les développeurs le testeurs doit être informer de cette modification afin qu'un test d'integration soit mis à disposition pour tester la nouvelle fonctionnalité du code.

6. Gestion des anomalies

En cas d'échec d'un test, le testeur doit évaluer l'importance de l'anomalie grâce à la priorité des cas d'utilisation présents dans la spécification des besoins. Ensuite il est nécessaire de prévenir l'équipe de développement afin de résoudre le problème. Le cas de test doit être entièrement revu, et un nouveau test devra avoir lieu.

Le testeur se devra d'être le plus précis possible lorsqu'il fera un retour au développeur, afin de faciliter la tâche de correction et de cibler rapidement l'origine du problème.

Chaque erreur détectée doit être notifiée dans le tableau des procédures (ci-après) et ne doit jamais disparaitre. Lorsqu'un nouveau test est effectué il faut ajouter une ligne pour indiquer l'évolution du problème jusqu'à sa résolution.

Non testé : veut dire que le test n'est pas encore realiser (concerne la partie sémantique ou le compilateur)

Procédures de test

Objet testé : **Utilisation du compilateur**			***version : 1.01***	
Objectif de test : **Compiler une application en mode monolithique**				
Procédure n° 2				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Créer des fichiers sources respectant la syntaxe kawa.	Fichier sources créés respectant la syntaxe.	EF_2	***ok***
2	Fournir au compilateur des fichiers sources respectant la syntaxe kawa sans utilisation de bibliothèques externes.	Avoir un exécutable unique après pré-compilation.		***non teste***
3	Fournir au compilateur des fichers sources ne respectant pas la syntaxe kawa sans utilisation de bibliothèque externes	Signaler un erreur et doit arrêter la compilation		***Non teste***

Objet testé : **Compilateur**			***Version : 1.01***	
Objectif de test : **Vérifier la concordance du nom des fichiers sortants**				
Procédure n° 8				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Choisir un nom spécifique pour les fichiers avant de compiler. Lancer la compilation.	Nom du fichier identique au nom choisi avant la compilation.	EXR_1	***Non teste***

Objet testé : **Compilateur**			***Version : 1.01***	
Objectif de test : **Grammaire KAWA respectée**				
Procédure n° 9				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Fournir des fichiers de code source au compilateur (respectant la grammaire)	En cas de respect de la grammaire KAWA la compilation doit se réaliser avec succès, sans signalement d'erreur.	EXR_2	***ok***
2	Fournir des fichiers de code source au compilateur (ne respectant pas la grammaire)	En cas de non-respect de la grammaire KAWA, le compilateur doit afficher une erreur et la positionner dans le fichier source. Si possible une solution pourra être proposée par le compilateur.		***ok***

Objet testé : **Compilateur**			***Version : 1.01***	
Objectif de test : **Utilisation correcte des mots clés**				
Procédure n° 10				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Utilisez des mots clés prédéfinis à des actions de développement spécifiques (« if », « while », « public » etc…) comme des noms de variables /fonctions /classes…	Le compilateur doit indiquer une erreur en indiquant l'endroit et la raison de l'erreur.	EXR_3	***Ok***
2	Utilisez tous les mots clés prévus par la grammaire et vérifier leur bon fonctionnement.	La compilation doit réussir et chaque mot clé doit fonctionner comme prévu dans la grammaire.		***ok***

Objet testé : **Compilateur**			***Version : 1.01***	
Objectif de test : **Format du fichier de sortie**				
Procédure n° 11				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Mauvaise extension du fichier de sortie.	-Erreur : le compilateur ne va pas reconnaître le fichier -les fichiers exécutables ne pourront plus utilisés les informations contenues dans le fichier pour monter en mémoire les références des bibliothèques externes.	EXR_4	***Non teste***
2	Non définition de son paquet par un élément.	Pas d'erreur : il doit être compilé dans le dossier dans lequel il se trouve.		***Non teste***

Objet testé : **Compilateur**			***Version : 1.01***	
Objectif de test : **Résolution de nom de classe**				
Procédure n° 13				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Appeler des méthodes utilisées dans des classes externes sans mettre le chemin entier mais en utilisant un import en début de fichier.	Le compilateur doit retrouver le chemin complet automatique grâce à l'import. Il ne doit donc pas y avoir d'erreur.	EXR_6	***Non teste***
2	Créer deux fichiers externes ayant une signature de méthode identique. Tenter d'appeler cette méthode dans le fichier principal (main)	Le compilateur doit indiquer une erreur et spécifier que dans un tel cas, malgré les imports, il faut mettre le chemin entier pour appeler la méthode adéquate.		***Non teste***

Objet testé : **Compilateur**			***Version : 1.01***	
Objectif de test : **Bon fonctionnement des paquets (package)**				
Procédure n° 14				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Créer des classes différentes au sein du même package et de package différents. Appeler des méthodes provenant de différentes classes en incluant le mot « package NomDuFichier » au début.	Le compilateur doit trouver de lui-même l'arborescence totale de chaque méthode (sauf si conflit).	EXR_7	***Non teste***

Objet testé : **Compilateur**			***Version : 1.01***	
Objectif de test : **Espace de nommage de classe**				
Procédure n° 15				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Créer deux fichiers portant le même nom au sein d'un même paquet.	Cela ne doit pas être réalisable, le compilateur doit donc indiquer l'erreur.	EXR_8	***Non teste***
2	Créer deux paquets différents et dans chacun d'eux créer un fichier ayant le même nom.	A l'inverse ici il ne doit pas y avoir de problème, les espaces de nommage entre paquet sont différents ; cette action est donc valide. Le compilateur ne doit pas annoncer d'erreur.		***Non teste***

Objet testé : **Compilateur**			***Version : 1.01***	
Objectif de test : **Gestion et création de classe concrète**				
Procédure n° 16				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Créer une classe dans un fichier à l'aide du mot clé « class ».	- Si le fichier porte le même nom que la classe concrète, un fichier .klass doit être généré lors de la compilation. - Sinon une erreur doit être renvoyée par le compilateur.	EXR_9	***Non teste***
2	Vérifier le bon fonctionnement de la classe en créant des attributs, des méthodes, des constructeurs et en les utilisant.	S'il n'y a pas d'erreur dans le code, le compilateur doit fonctionner et faire sortir un exécutable fonctionnel.		***Non teste***

Objet testé : **Compilateur**	***Version : 1.01***			
Objectif de test : **Création et gestion de classe abstraite**				
Procédure n° 17				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Créer une classe abstraite à l'aide des mots clés « abstract class ».	- Si le fichier porte le même nom que la classe abstraite, un fichier .klass doit être généré lors de la compilation. - Sinon une erreur doit être renvoyée par le compilateur.	EXR_10	***Non teste***
2	Déclarer un objet avec pour type la classe abstraire et instancier cet objet avec une classe concrète fille.	Le compilateur ne doit pas retourner d'erreur et doit générer correctement le fichier.		***Non teste***
3	Instancier un objet de cette classe.	Cette action ne doit pas fonctionner, le compilateur doit retourner une erreur.		***Non teste***
4	Créer une classe concrète et signer une méthode de cette classe avec le mot clé « abstract ».	-Le compilateur doit vérifier si cette classe contient le mot « abstarct » sinon rend cette derniere abstraite – Ou demander à l'utilisateur de mettre le mot clé « abstract » devant la classe concernée		***Non teste***
5	Ecrire des méthodes, des attributs et des constructeurs.	Les classes abstraites peuvent contenir des méthodes, attributs et constructeurs. Aucune erreur ne doit ressortir.		***Non teste***
6	Ecrire le corps d'une méthode signé « abstract ».			***Non teste***

Objet testé : **Compilateur**	***Version : 1.01***			
Objectif de test : **Création et gestion d'interface**				
Procédure n° 18				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Créer une interface à l'aide du mot clé « interface ».	-Si le fichier porte le même nom que l'interface, un fichier .klass doit être généré lors de la compilation. - Sinon une erreur doit être renvoyée par le compilateur.	EXR_11	***Non teste***
2	Déclarer un objet avec pour type la classe abstraire et instancié le avec une classe concrète fille.	Le compilateur ne doit pas retourner d'erreur et doit générer correctement le fichier.		***Non teste***
3	Instancier un objet de cette classe	Cette action ne doit pas fonctionner, le compilateur doit retourner une erreur.		***Non teste***
4	Créer des constructeurs, méthodes et attributs.	Cette action ne doit pas être valide. Une erreur doit être retournée.		***Non teste***
5	Créer des prototypes de méthodes	- Si la visibilité d'un prototype est différente de public, une erreur doit être renvoyée. - Sinon la compilation se fait avec succès.		***Non teste***

Objet testé : **Compilateur**	***Version : 1.01***			
Objectif de test : **Dérivations et implémentations**				
Procédure n° 21				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	*Héritage multiple « indirect » :* Créer plusieurs classes mères/filles (héritages) en essayant d'appeler les méthodes / fonctions de ses parents et enfants. Ex : A -> B -> C -> D	- Toutes les méthodes et attributs des classes mères ayant pour visibilité « protected » ou « public » doivent être accessibles directement depuis la classe fille. - Les classes mères ne doivent pas pouvoir accéder aux méthodes et attributs de leurs classes filles.	EXR_14	***Non teste***

2	*Héritage multiple « direct » :* Créer une classe fille qui hérite de deux classes (ex : A -> B et A -> C)	Le compilateur ne doit pas autoriser cette action et doit déclencher une erreur.		**Non teste**
3	*Implémentations multiples « directes » :* Créer une classe qui hérite d'une puis de plusieurs interfaces.	Le compilateur ne déclenche pas d'erreur si la classe implémente les méthodes de toutes les interfaces qu'elle dérive.		**Non teste**
4	*Héritage d'interface/classe abstraites :* Créer une interface et une classe abstraite. Faire hériter une classe fille de ces deux classes.	- Si toutes les méthodes de l'interface et toutes les méthodes « abstract » de la classe abstraite sont implémentées dans la classe fille la compilation doit se dérouler avec succès. - A l'inverse si l'une d'entre elles n'est pas implémentée le compilateur doit indiquer une erreur.		**Non teste**

Objet testé : Compilateur	**Version : 1.01**			
Objectif de test : Mécanisme de constructeur				
Procédure n° 22				
N°	**Actions**	**Résultats attendus**	**Exig.**	**OK/ NOK**
1	Créer une classe sans constructeur et instancier un objet.	Kawac doit fournir par défaut un constructeur, l'instanciation doit donc se faire sans soucis, avec les attributs initialisés à leur valeur par défaut.	EXR_15	**A revir**
2	Créer un ou plusieurs constructeurs (avec des arguments différents) et instancier des objets en appelant chacun d'entre eux.	Chaque instanciation doit se faire avec la bonne initialisation des variables. Aucune erreur ne doit apparaitre.		**Non teste**

Objet testé : Compilateur	**Version : 1.01**			
Objectif de test : Reconnaissance et définitions de méthodes et de variables				
Procédure n° 24				
N°	**Actions**	**Résultats attendus**	**Exig.**	**OK/ NOK**
1	Créer des méthodes et initialiser des variables locales au sein de ces méthodes. Utiliser ces variables dans d'autres méthodes.	Le compilateur doit indiquer une erreur. Une variable étant limitée à une zone d'action, une variable locale est limitée à son bloc de création.	EXR_17	**Non teste**

2	Créer des variables de classe et les utiliser dans chaque méthode de la classe.	Aucune erreur ne doit survenir, une variable de classe est accessible partout au sein d'une classe. En revanche on ne doit pas pouvoir les utiliser hors de la classe.		***Non teste***

Objet testé : **Compilateur**		***Version : 1.01***		
Objectif de test : **Reconnaissance et définition d'attribut d'objet**				
Procédure n° 25				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Création d'attribut	Attribut créé.	EXR_18	***ok***
2	Utiliser l'attribut au sein de différentes méthodes de la classe.	L'attribut doit être accessible partout au sein de la classe. Aucune erreur.		***ok***
3	Faire un appel à cet attribut depuis une autre classe, ou depuis l'extérieur à sa classe d'appartenance.	-Si l'attribut est déclaré public il doit être accessible (il peut être nécessaire de le préfixer par le nom de la classe)		***Non teste***
4	Faire un appel à cet attribut depuis une autre classe, ou depuis l'extérieur à sa classe d'appartenance.	- Si l'attribut n'est pas public il ne doit pas être accessible (exception faites de l'héritage et de la visibilité protected).		***Non teste***

Objet testé : **Compilateur**		***Version : 1.01***		
Objectif de test : **Définition d'attributs statiques**				
Procédure n° 26				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Création d'attribut statique	Attribut créé.	EXR_19	***ok***
2	Appel de l'attribut statique par une méthode.	-Si la méthode est statique : Pas d'erreur. -Si la méthode est non statique : Erreur.		***Non teste***
3	Création de méthode statique	Méthode créée.		***ok***

4	Appel de l'attribut non static par une méthode static	-Si l'attribut est déclaré hors de la méthode static une erreur doit être lancé -Si l'attribut est déclaré dans la méthode static pas d'erreur		***Non teste***

Objet testé : **Compilateur**			***Version : 1.01***	
Objectif de test : **Définition d'attributs de constantes**				
Procédure n° 27				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Création de l'attribut constante avec initialisation.	Constante créée.	EXR_20	***Non teste***
2	Création de l'attribut constante sans initialisation.	Erreur : la constante doit être initialisée.		***Non teste***
3	Modification de la constante.	Erreur : la constante ne peut être modifiée.		***Non teste***

Objet testé : **Compilateur**			***Version : 1.01***	
Objectif de test : **Définition d'attributs final**				
Procédure n° 28				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Création d'attribut final	Attribut créé.	EXR_19	***ok***
2	Redéfinition de l'attribut final	Erreur l'attribut final ne peut etre redéfinie		***Non teste***

Objet testé : **Compilateur**	***Version : 1.01***			
Objectif de test : **Définition de méthode finale**				
Procédure n° 31				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Création de la méthode.	Méthode créée.	EXR_24	***ok***
2	Redéfinition de la méthode.	Erreur : la méthode ne peut être redéfinie		***Non teste***
3	Surcharge de la méthode.	Erreur : la méthode ne peut être surchargée.		***Non teste***

Objet testé : **Compilateur**	***Version : 1.01***			
Objectif de test : **Définition de méthode statique**				
Procédure n° 32				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Création de la méthode.	Méthode créée	EXR_25	***ok***
2	Appel d'un attribut non statique dans la méthode.	Erreur : ne peut appeler que des attributs statiques		***Non teste***
3	Appel de la méthode avec le nom d'une autre classe que celle dans laquelle elle est définie.	Erreur : elle ne peut être appelée qu'avec le nom de la classe dans laquelle elle a été déclarée.		***Non teste***

Objet testé : **Compilateur**	***version : 1.01***			
Objectif de test : **Héritage**				
Procédure n° 33				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Création de méthode ou d'attribut	Méthode ou attribut créé.	EXR_26, EXR_27	***ok***
2	L'utilisateur fait hériter la méthode ou l'attribut créé à une autre méthode ou attribut existant.	Héritage de la méthode ou de l'attribut existant par la méthode ou l'attribut créé.		***Non teste***

'Objet testé : **Compilateur**	***version : 1.01***			
Objectif de test : **Redéfinition de méthode**				
Procédure n° 34				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Création de méthode.	Méthode créée.	EXR_28	***ok***
2	Redéfinition de la méthode avec des paramètres différents	Erreur : la méthode redéfinie doit avoir les mêmes paramètres que celle dont elle hérite.		***Non teste***
3	Redéfinition de la méthode et renvoyant un type différent.	Erreur : la méthode doit avoir le même type de retour que celle dont elle hérite.		***Non teste***

Objet testé : **Compilateur**	***version : 1.01***			
Objectif de test : **Surcharge de la méthode**				
Procédure n° 35				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Création de méthode.	Méthode créée.	EXR_29	***ok***
2	Définir une méthode avec le nom d'une autre existante et renvoyant des types de valeurs différentes.	Erreur : les types de retours doivent avoir les mêmes types de valeurs.		***Non testé***
3	Définir une méthode avec le nom d'une autre existante et ayant les même paramètres.	Erreur : elle doit avoir des paramètres différents.		***Non teste***

Objet testé : **Compilateur**			***version : 1.01***	
Objectif de test : **Concept de visibilité**				
Procédure n° 36				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Création d'attribut ou de méthode de type private ou public ou protected.	Variable ou attribut créé.	EXR_30	***ok***
2	Type : private Type : protected Type : public	- Ne peut être appelé en dehors de la classe - Ne peut être appelé en dehors de la classe sauf pour les hérités de la classe interface. - Autoriser l'accès à tous.		***Non teste***

Objet testé : **Compilateur**			***version : 1.01***	
Objectif de test : **Portée de variable et résolution de noms de variables**				
Procédure n° 37				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Création de variable dans un bloc d'instruction.	Variable créée.	EXR_31	***ok***
2	Appel de la variable en dehors du bloc où elle a été déclarée.	Si c'est une variable globale : pas d'erreur Sinon : elle reste locale au bloc dans lequel elle a été déclarée, donc donnera une erreur.		***Non testé***

Objet testé : **Compilateur**	***version : 1.01***
Objectif de test : **Reconnaissance des expressions conditionnelles**	
Procédure n° 39	

N°	*Actions*	*Résultats attendus*	*Exig.*	*OK/ NOK*
1	Création de l'expression conditionnelle.	Expression créée.	EXR_33	*ok*
2	Mauvaise implémentation de la condition d'entrée dans l'expression.	Erreur : l'expression conditionnelle ne sera pas exécutée.		*ok*
3	Mauvaise fermeture des accolades pour montrer le début et la fin de l'expression conditionnelle.	Erreur : plantage ou envoie d'un message d'erreur précisant le manque d'une accolade.		*ok*

Objet testé : **Compilateur**	***version : 1.01***
Objectif de test : **Reconnaissance des expressions de bouclage**	
Procédure n° 40	

N°	*Actions*	*Résultats attendus*	*Exig.*	*OK/ NOK*
1	Création d'une boucle.	Boucle créée.	EXR_34	*ok*
2	Définition d'un bloc de boucles dans un autre.	Pas d'erreur de compilation, un bloc de boucle peut en contenir d'autres.		*ok*
3	Mauvaise implémentation de la condition d'entrée dans l'expression.	Erreur : l'expression conditionnelle ne sera pas exécutée.		*Non testé*
4	Oubli de la condition d'arrêt de la boucle.	Erreur : boucle infinie et non-exécution de la suite du programme.		*Non ok*

Objet testé : **Compilateur**			***version : 1.01***	
Objectif de test : **Bloc de classe**				
Procédure n° 41				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Définition d'un bloc de classe sans les { }	Erreur : la classe ne sera pas reconnue par le compilateur.	EXR_35	***ok***
2	Déclaration d'une variable liée à cette classe mais qui est faite en dehors du bloc de classe.	Erreur : la variable ne sera pas reconnue par le compilateur qui retourne une erreur.		***ok***
3	Déclaration d'une classe imbriquée.	Erreur : le compilateur va « planter ».		***ok***

Objet testé : **Compilateur**			***version : 1.01***	
Objectif de test : **Bloc d'instructions**				
Procédure n° 42				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Création d'un bloc d'instructions	Bloc créé.	EXR_36	***ok***
2	Définition d'un bloc d'instruction dans un autre.	Pas d'erreur.		***ok***
3	Définition d'un bloc d'instruction sans { }	Erreur : le compilateur renvoie une erreur.		***ok***

Objet testé : **Compilateur**			***version : 1.01***	
Objectif de test : **Reconnaissance des commentaires**				
Procédure n° 43				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Mauvaise définition d'un commentaire.	La ligne de commentaire sera interprétée comme une instruction ce qui entrainera donc une erreur dans le programme.	EXR_37	***ok***
2	Oubli de la balise fermante du commentaire.	Toute la suite d'instructions après la balise du commentaire sera en commentaire.		***ok***
3	Pouvoir mettre des commentaires sur plusieurs lignes	Doit pas prendre en compte toutes les instructions contenu dans le commentaire		***Non ok***

Objet testé : **Compilateur**			***version : 1.01***	
Objectif de test : **Message d'erreur explicite**				
Procédure n° 44				
N°	***Actions***	***Résultats attendus***	***Exig.***	***OK/ NOK***
1	Laissez volontairement des erreurs dans les fichiers sources à des endroits spécifiques.	Vérifier que les messages indiquant les erreurs soient clairs, compréhensifs et correspondent à l'erreur.	EQ_1	***ok***

Non testé : veut dire que le test n'est pas encore realiser (concerne la partie sémantique ou le compilateur)

STRATEGIE DE REALISATION DES TESTS

Afin de voir si la grammaire de l'AST marche correctement j'ai prévue de faire certains tests.

- Il y a ura un fichier **test1** qui sera le fichier d'entrée à donner à la grammaire et qui vas contenir des déclarations de variables,de tableaux, des affections, et aussi des calculs. Il contiendra aussi des erreurs afin de voir si ces derniers seront detecté. On aura donc une sortie qui doit correcpondre donc aux résultats escomptés sinon cela veut dire que la grammaire ne marche pas .

- Il y aura aussi un fichier nommée **test2** qui lui vas contenir la déclaration d'une classe X. Le but est de vérifier si la syntaxe utilisée pour la définition de la classe X est correcte elle contiendra elle aussi evidement des erreurs de syntaxe afin de voir si la grammaire pourra les detecter et afficher.

- Ensuite il y aura d'autres fichiers tests qui vont tester d'autre fonctionnalités telque l'héritage, le polymorphisme , les interface, la surcharge de méthode etc.... Ces fichiers contiendront eux aussi des erreurs afin de vérifier la bonne marche de l'analyseur syntaxique.

En ce qui concerne l'**Analyseur sémantique** des erreurs sémantiques seront introduites dans les fichiers tests ayant permis de faire les tests syntaxiques afin de voir si elles seront détecter par l 'analyseur sémantique.

Ces tests seront consideré comme des tests unitaires et il y a aura bien sûre d'autres tests comme des tests de non regression en cas d'ajout de fonctionnalité ou de modification d'un de ces modules et des tests d'intégration afin de tester un livrable. Cela nous permettra donc de mettre à jour notre cahier de recette , et nous permettra ainsi de savoir quelles sont les fonctionnalités de notre compilateur LLVM qui marchent et celles qui ne marchent pas.