UNIVERSITE SIDI MOHAMED BEN ABDELLAH
FACULTÉ DES SCIENCES ET TECHNIQUES FÈS
DÉPARTEMENT D'INFORMATIQUE

FST FES

PROJET DE FIN D'ETUDES

MASTER SCIENCES ET TECHNIQUES
SYSTÈMES INTELLIGENTS & RÉSEAUX

DÉTECTION, LOCALISATION ET CORRECTION AUTOMATIQUE D'ERREURS DANS UN PROGRAMME C POUR AMÉLIORER L'ÉVALUATION AUTOMATIQUE BASÉE SUR L'ANALYSE DYNAMIQUE

LSIA

LIEU DE STAGE : LABORATOIRE SYSTÈMES INTELLIGENTS ET APPLICATIONS
L.S.I.A DE LA FACULTÉ DES SCIENCES ET TECHNIQUES

RÉALISÉ PAR :

Ismail NAIT ABDELLAH OUALI

SOUTENU LE : 20 JUIN 2016

ENCADRÉ PAR :
M. BEN ABBOU RACHID
M. ZAHI AZEDDINE
MME. MERNISSI ARIFI SARA

DEVANT LE JURY COMPOSÉ DE :
PR. BEN ABBOU RACHID
PR. ZAHI AZEDDINE
PR. BEGDOURI AHLAME
PR. ZARGHILI ARSALANE

ANNÉE UNIVERSITAIRE 2015-2016

Résumé

L'évaluation automatique de programmes par la méthode d'analyse dynamique consiste à exécuter le programme et à comparer les résultats de l'exécution aux sorties attendues.

Cette méthode présente un inconvénient majeur qui se résume dans l'incapacité à évaluer un programme contenant des erreurs de compilation ou d'exécution. Ainsi, dans le cas de l'échec de la compilation ou de l'exécution du programme évalué, le processus de l'évaluation de celui-ci est interrompu à cette étape.

Ce projet consiste à proposer une méthode de détection, de localisation et de correction automatique d'erreurs dans le programme évalué. Ce traitement préliminaire est utilisé dans le but de garantir l'exécution et l'évaluation automatique du programme par la méthode d'analyse dynamique.

Mots clés :

Analyse dynamique, localisation, correction d'erreurs, évaluation de programmes

Abstract

Dynamic analysis consists of executing a program using a set of test-cases. The outputs of these executions are matched against the expected ones; outputs generated by the execution of a model program using the same test-cases.

Dynamic analysis method presents a major limit that mainly revolves around the inability to analyze a program in the case of its compilation or/and execution failure. Thus the evaluation process is interrupted at that stage.

In this work, we propose a method that consists of detecting, localizing and fixing automatically errors that exist in the assessed program. It is a preprocessing treatment used with the aim to maximize chances for the evaluated program to be executed and then assessed using the dynamic analysis method.

Key words:

Dynamic analysis, error localizing, error fixing, program assessment

Dédicace

A la mémoire de mon père

Aucune dédicace ne saurait exprimer mon respect, mon amour éternel et ma considération pour les sacrifices que tu avais consenti pour mon instruction et mon bien-être. J'espère que ta bénédiction m'accompagne toujours et que Dieu ait ton âme dans sa sainte miséricorde.

A ma mère

Je te remercie pour tout le soutien et l'amour que tu me porte depuis mon enfance et je te dédie ce travail avec tous mes vœux de bonheur, de santé et de bien-être.

A mes sœurs et mon frère

Votre soutien et encouragement me marqueront à jamais.

Je vous exprime à travers ce travail mes sentiments de fraternité et d'amour.

A mon encadrante MERNISSI ARIFI Sara

Un remerciement particulier et sincère pour tous vos efforts fournis et votre valeureuse orientation.

Que ce travail soit un témoignage de ma gratitude et de mon profond respect.

Remerciements

Je tiens à exprimer mes remerciements, mon respect et ma gratitude à tous mes encadrants, M. BENABBOU Rachid, M. ZAHI Azeddine et Mme. MERNISSI ARIFI Sara. Merci infiniment, sans votre support ce travail n'aurait pas pu voir le jour.

Mes plus vifs remerciements s'adressent aussi à tous mes chers professeurs du département informatique de la faculté des sciences et techniques de Fès pour leurs efforts inégaux afin de nous garantir une formation de qualité.

Je voudrais également exprimer ma reconnaissance envers les amis et collègues qui m'ont apporté leur support moral et intellectuel tout au long de ma démarche.

Sans oublier de remercier tous les membres du jury qui ont agréé d'évaluer et de juger cet humble travail.

Finalement, je présente mes remerciements, à toute personne ayant contribué de près ou de loin à l'élaboration de ce travail.

Table des Matières

Table des figures

Liste des abréviations

APAS : Automated Programming Assessment Systems
[Système d'Evaluation Automatisé des Programmes]

CLAAS : C Language Automatic Assessment System

Introduction

Ce travail a été effectué dans le cadre d'un projet de fin d'études au sein du laboratoire Systèmes Intelligents et Applications L.S.I.A de la Faculté des Sciences et Techniques de Fès.

Le projet a été suggéré dans le cadre d'une thèse de doctorat intitulée : « l'évaluation automatique de programme, étude de cas de la programmation en langage C » préparée par Mme. MERNISSI ARIFI Sara au sein du même laboratoire.

L'évaluation est une pratique pédagogique essentielle à l'apprentissage. Elle permet à l'enseignant mais surtout à l'apprenant de repérer les types d'erreurs commises. Ainsi, l'évaluation a pour premier but de signaler aux formateurs le niveau et la qualité d'acquisition des connaissances par les étudiants. L'évaluation permet en outre de motiver les sujets apprenants, de les inciter à se dépasser au-delà de leurs limites.

D'autre part, l'évaluation permet au formateur de remettre en cause ses méthodes de travail, de les réviser et de les adapter à la formation tant du point de vue contenu que du point de vue de la forme.

Comme dans toute autre discipline, dans le cours de programmation, l'évaluation de programmes soumis par les étudiants est une étape primordiale qui informe sur le degré d'acquisition des théories et des techniques appris durant le cours. Cependant, l'évaluation manuelle de programmes s'avère couteuse en terme de temps, vu le nombre de plus en plus grand des étudiants mais également à cause de l'intégration de cette discipline dans pratiquement toutes les branches scientifiques. Cela rend la tâche d'évaluation de plus en plus fastidieuse et implique la diminution du nombre de tests programmés par ces derniers. S'ajoute à cela plusieurs facteurs qui peuvent influencer l'évaluateur humain à savoir la subjectivité, la fatigue, le favoritisme, etc. [1] Cela affecte amplement les résultats de l'évaluation mais aussi la qualité du feed-back fournis à l'étudiant. De ce fait, le recours à l'évaluation automatique est justifié.

L'évaluation automatique de programme a constitué depuis longtemps une thématique à laquelle on attache beaucoup d'importance de la part des institutions académiques, mais également de la part des sociétés commerciales conscientes de la valeur ajoutée de l'utilisation des systèmes d'évaluation automatique de programme.

L'évaluation automatique de programmes présente des avantages énormes pour les deux principaux intervenants du processus d'apprentissage : l'étudiant et l'enseignant. D'une part, l'étudiant profite de plus d'exercices lui permettant de pratiquer ses acquis, mais également du feed-back instantané et détaillé renvoyé par le système d'évaluation automatique de programmes. D'autre part, l'enseignant peut exploiter les résultats et les informations produits afin de cerner les faiblesses qui peuvent être relatives à plusieurs variantes et

proposer, par la suite, les actions et les mesures nécessaires qui visent à soulever les difficultés constatées.

Les différents systèmes d'évaluation automatique de programmes existants sont fondés sur deux principales approches d'analyse : la méthode dynamique et la méthode statique.

Dans le cadre de ce travail de recherche, nous présentons un système d'évaluation automatique de programmes innovant, basé sur la méthode d'analyse dynamique améliorée. L'amélioration concerne la limite de cette méthode relative à son incapacité à analyser un programme qui ne se compile et/ou ne s'exécute pas.

Ce projet consiste à proposer une méthode de détection, de localisation et de correction automatique d'erreurs dans le programme évalué, le but étant de garantir l'exécution et l'évaluation automatique du programme par la méthode d'analyse dynamique même quand celui-ci contient des erreurs de compilation ou d'exécution.

…

Présentation du laboratoire :

Le laboratoire SIA, crée en 2011, est une unité de Recherche du Centre d'Etudes Doctorales en Sciences et Techniques de l'Ingénieur domicilié à la Faculté des Sciences et Techniques de Fès et regroupant des laboratoires de recherche tous accrédités par l'Université Sidi Mohamed Ben Abdellah de Fès, et domiciliés à la Facultés des Sciences et Techniques, l'Ecole Supérieure de Technologie, la Faculté Polydisciplinaire de Taza, l'Ecole Nationale des Sciences Appliquées de Fès et l'ENS de Fès.

Le LSIA est composé de 12 enseignants-chercheurs du département d'Informatique de la FST de Fès et de 23 doctorants. Cette imbrication étroite entre enseignement et recherche, est un élément essentiel de la dynamique du laboratoire.

Les thématiques de recherche se situent au cœur des Sciences et Technologies de l'Information et de la Communication et s'articulent essentiellement autour des thématiques de recherche des enseignants chercheurs du laboratoire et assure une large couverture thématique présentant un atout très important pour le laboratoire.

Chapitre1. Etat de l'art des systèmes d'évaluation automatique de programmes

I. Les systèmes d'évaluation automatique de programmes

L'utilisation de systèmes d'évaluation automatique de programme provoque, comme à chaque fois que la technologie semble se substituer à l'humain, une réflexion sur le sens et la valeur de la fonction qu'il va remplir. Nous voulons dire par Un système d'évaluation automatisé des programmes (APAS, acronyme de : Automated Programming Assessment Systems), un système automatique permettant de classer, d'évaluer et de vérifier l'exactitude des exercices de programmation des étudiants. L'utilisation d'un tel système rend possible :

- L'autoévaluation par les étudiants eux-mêmes.
- L'analyse des erreurs par les étudiants et par les enseignants

L'utilisation de système d'évaluation automatique de programme permet à l'étudiant de recevoir une note reflétant le degré de réalisation des objectifs. Cette note est justifiée par le feed-back instantané renvoyé à l'étudiant pour l'aider à identifier ses lacunes et à progresser à travers l'utilisation régulière de ces outils. Les résultats produits par l'évaluation automatique de programmes écrits par les étudiants constituent un élément essentiel dans le tableau de bord de l'enseignant, et ce à travers l'analyse des résultats et des erreurs renseignant sur le degré d'achèvement des objectifs visés par l'enseignant tout au long du cours.

L'utilisation des systèmes d'évaluation automatique de programme a fait preuve d'un grand intérêt dans plusieurs disciplines informatiques, telles que le domaine de vérification et test de logiciel, la compréhension de programme, etc. Dans le contexte éducatif et académique, l'utilisation de ces outils est de plus en plus sollicitée.

Les systèmes d'évaluation automatique de programme existants sont classifiés selon l'approche optée pour l'évaluation du programme soumis. Les deux approches utilisées sont : l'analyse statique et l'analyse dynamique, à noter aussi l'existence de systèmes hybrides qui combinent des méthodes appartenant aux deux approches.

La section suivante est dédiée à la présentation de ces deux méthodes, de leurs avantages et de leurs limites.

II. Les méthodes d'analyse de programmes

L'analyse statique et l'analyse dynamique se présentent comme étant les deux approches sur lesquelles se basent les systèmes d'évaluation automatique des programmes, chacune ayant ses avantages et ses limites.

1) La méthode d'analyse statique

L'analyse statique permet de recueillir des informations sur le programme sans avoir à l'exécuter et par conséquent, d'éliminer les risques liés à l'exécution de programme. Plusieurs méthodes et techniques ont été conçues dans le cadre de cette approche telles que [4]:

- **L'analyse du style de programmation** : consiste à évaluer la lisibilité du programme et donc sa qualité en considérant certaines propriétés du programme telles que la présence de commentaires, les noms des variables, l'usage de constantes, la portée des variables, etc.
- **L'analyse des métriques** : consiste à mesurer les propriétés d'un programme. On peut à travers ces métriques jauger la complexité et la fiabilité du code, la qualité du processus de développement et la performance du programme une fois terminé.
- **L'analyse par mots-clés** : qui consiste à rechercher des mots-clés définis par l'évaluateur dans le corps du code source du programme évalué.
- **L'analyse structurelle et non structurelle**: qui consiste à comparer le programme à évaluer à un ou à plusieurs autres programmes dits "experts" ou "modèles" fourni par l'évaluateur. La note attribuée au programme est relative au degré de similarités détectées entre le programme et la solution.

L'avantage majeur de cette méthode est la capacité à analyser et évaluer même un programme contenant des erreurs, puisqu'on n'a pas recourt à son exécution. Cependant, vu sa complexité, peu de projets se basant exclusivement sur cette approche ont vu le jour.

2) La méthode d'analyse dynamique

L'analyse dynamique consiste à exécuter le programme à évaluer avec un ensemble de test-cases appelés aussi cas de tests, et de comparer les résultats produits avec les sorties attendues. Un test-case est composé des entrées et de l'ensemble de sorties résultant de l'exécution d'un programme solution en utilisant ces entrées. Si les sorties de l'exécution du programme avec les entrées d'un test-case sont identiques aux sorties attendues de celui-ci, le programme est alors considéré correct. Cette approche est adoptée par la majorité des systèmes d'évaluation de programme existants.

La méthode d'analyse dynamique s'intéresse à la sémantique opérationnelle du programme évalué, selon laquelle, on associe à un programme une suite d'états d'une machine (abstraite) qui l'exécute, on va donc observer les traces d'exécution du programme.

Selon la méthode d'analyse dynamique, un programme est dit correct s'il produit les mêmes traces d'exécution que le programme expert ; tous deux s'exécutant sur les mêmes données;

c'est à dire que le programme évalué et le programme expert sont équivalents du point de vue sémantique opérationnelle.

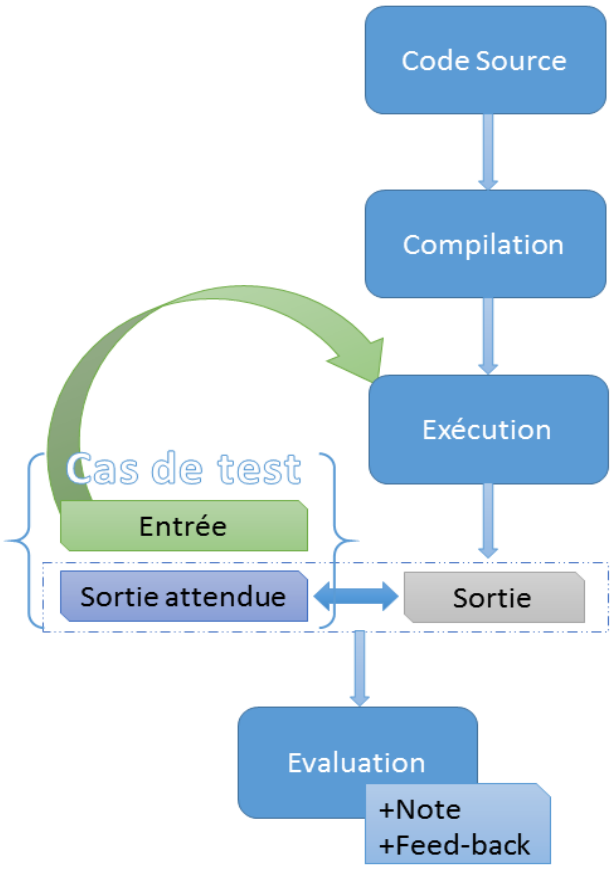

Figure 1 : Processus d'analyse dynamique

En analyse dynamique, on peut distinguer différentes méthodes pour évaluer un programme :

- **Le black-boxing** qui considère le programme comme étant une seule entité. Dans ce cas ce sont les sorties de l'exécution du programme entier qui sont examinées. Ainsi, le jugement donné par rapport au programme ne peut être que "correct" ou "incorrect".
- **Le grey-boxing** qui permet d'évaluer les sorties générées par l'exécution de chaque fonction appartenant au programme. Cette méthode résout partiellement le problème de la note donnée qui se répartit sur l'ensemble des fonctions constituant le programme selon un pourcentage fixé [2].

L'avantage majeur de la méthode d'analyse dynamique réside dans la facilité de son implémentation. Cela justifie son adoption par la plupart des systèmes automatiques d'évaluation de programmes. Néanmoins, elle présente quelques limites. L'échec de la

compilation d'un programme entrave son exécution et son évaluation par la méthode dynamique qui se base sur les sorties de l'exécution du programme évalué.

Dans le cas de l'échec de compilation du programme évalué, le feed-back ne peut être que la liste des erreurs que le programme contient. Dans ce cas, on constate une limite au niveau du feed-back [3].

Si le programme parvient à être compilé et par la suite exécuté et analysé, dans ce cas le feed-back contient les sorties des tests-cases échoués et les résultats attendus de ceux-ci. La localisation des erreurs commises est laissée à la charge de l'étudiant. Cela constitue une limitation au niveau du feed-back dont la plupart des systèmes d'évaluation automatique basés sur la méthode d'analyse dynamique souffrent.

Un autre inconvénient de la méthode d'analyse dynamique se présente dans l'incapacité à déterminer si le programme évalué est conforme aux exigences de l'évaluateur de point de vue utilisation de méthodes spécifiques, même quand ce programme réussit les test-cases, vu que l'analyse dynamique n'examine pas le code source du programme analysé, mais s'intéresse uniquement aux sorties de son exécution.

3) La méthode d'analyse hybride

La fusion des deux méthodes d'analyse de programme, à savoir les méthodes statique et dynamique, donne lieu à une troisième méthode appelée méthode d'analyse hybride. Cette méthode consiste à combiner la méthode dynamique à l'une des méthodes situées dans le cadre de l'approche statique tel que les systèmes basés sur la méthode d'analyse dynamique tout en intégrant l'analyse par mots-clés ou par métrique en tant que méthodes statiques dans l'évaluation de programme. Cette complémentarité entre les deux méthodes d'analyse de programme donne lieu à une synergie intéressante dans le sens où chaque méthode contribue à améliorer le processus de l'évaluation et la qualité du feed-back.

III. Historique des systèmes d'évaluation automatique de programmes :

Le recensement des projets réalisés et l'observation de leur succession et de leur évolution permet de repérer nettement les besoins et le manque à combler dans ce domaine.

Le besoin d'automatiser la tâche de l'évaluation de programme s'est manifesté il y a plusieurs décennies à travers les premières contributions dans ce domaine.

En 1960, le premier rapport sur les systèmes d'évaluation automatique a été publié dans le magazine CACM (Communications of the Association for Computing Machinery) par Hollingsworth [5] dans lequel il décrit l'expérience de l'utilisation du "Automatic grader" dans le "Rensselaer Polytechnical Institute" à New York.

Cinq ans après, Forsythe [6] a introduit un système qui se base sur un principe fondamental des systèmes d'évaluation automatique modernes en évaluant la solution soumise par la validation d'un ensemble de tests.

En 1969, le système BAGS [7] (Basser Automatic Grader Scheme) est développé à l'université de Sydney. La note est attribuée par le système selon cinq critères : la compilation réussie, l'exécution terminée, la validation du premier test, la validation du deuxième test et le temps de réalisation. Le programme pénalise toute tentative de soumission après la première. Le système peut fournir le compte rendu d'une classe par rapport à un exercice ou sur une période déterminée.

Plus tard, en 1988, Ceilidh [8] a vu le jour dans l'université de Nottingham et depuis, il a été adopté par plus de 200 instituts d'enseignement supérieur à travers 30 pays. L'implémentation d'un tel système a énormément influencé la recherche et l'implémentation d'autres systèmes d'évaluation automatique. Neuf ans plus tard, un descendant de Ceilidh a été implémenté sous le nom de CourseMaster [9], une version améliorée de l'ancien système en termes de facilité d'utilisation, de maintenabilité, d'expansibilité et de mécanisme de feed-back.

En 1994, Kassandra [10] a été développé à ETH Zurich. Ce système a été désigné pour évaluer les exercices de Maple et Matlab. L'évaluation est basée sur la validation des cas d'essai. Des fonctionnalités auxiliaires ont été fournies tel que l'accès par l'étudiant aux rapports de ses évaluations précédentes, et la possibilité d'interagir avec le système via un compte.

Ultérieurement, d'autres systèmes d'évaluation automatique orientés web ont été implémentés tels que BOSS, Web-Cat [11] plus robustes de point de vue sécurité et stratégie de feed-back et d'interopérabilité avec les LMS (Learning Management System).

Pour résumer, les systèmes d'évaluation automatique appartenant à l'ancienne génération produisaient les feed-back en se basant sur la comparaison des sorties du programme aux résultats escomptés. Dans les générations les plus récentes, d'autres aspects du programme à évaluer sont pris en considération.

Chapitre2. Conception et réalisation d'un système d'évaluation automatique de programmes

I. Conception et interfaces de CLAAS

1) Conception et fonctionnalités

Dans le cadre de ce projet de recherche, plusieurs systèmes d'évaluation automatique de programmes basés sur la méthode d'analyse dynamique ont été expérimentés, exemple : SHARIF JUDGE. Nous avons procédé par la suite à la conception de notre propre implémentation de cette méthode dans un système basé sur la méthode d'analyse dynamique de programme : CLAAS, acronyme de C Language Automatic Assessment System. CLAAS a été conçu dans le but de garantir des améliorations relativement à quelques points faibles relevés durant l'utilisation des systèmes expérimentés.

Le système proposé est développé dans le but de :

- Offrir aux étudiants un outil d'auto-évaluation.
- Retourner un feed-back instantané et détaillé suite à chaque évaluation.
- Epargner les enseignants de l'effort et du temps consacrés à la correction manuelle des programmes.
- Permettre aux enseignants de proposer un plus grand nombre d'exercices.

Le système réalisé est basé sur l'architecture « client/serveur ». Ses fonctions principales se résument dans :

- La gestion des classes
- La gestion des enseignants
- La gestion des étudiants
- La gestion des épreuves
- La gestion des soumissions

Afin d'accomplir ces différentes fonctions, plusieurs acteurs interviennent comme présenté dans les diagrammes de cas d'utilisation. Les cas d'utilisations permettent de structurer les besoins des utilisateurs et les objectifs correspondants d'un système. Ils centrent l'expression des exigences du système sur ses utilisateurs en clarifiant et en organisant leurs besoins (les modéliser). Les figures 2, 3 et 4 présentent les diagrammes de cas d'utilisation de CLAAS.

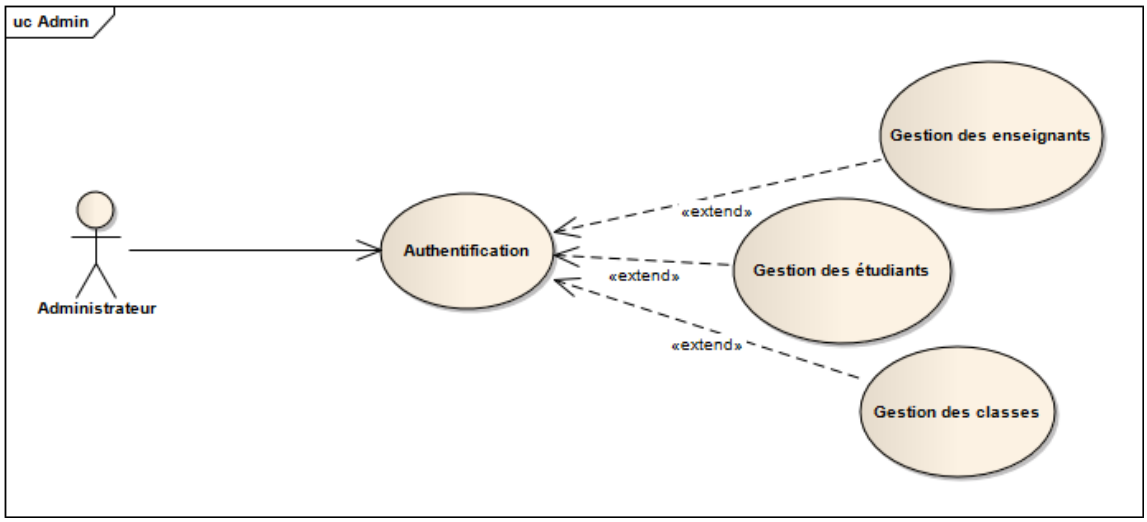

Figure 2: Diagramme de cas d'utilisation de l'administrateur

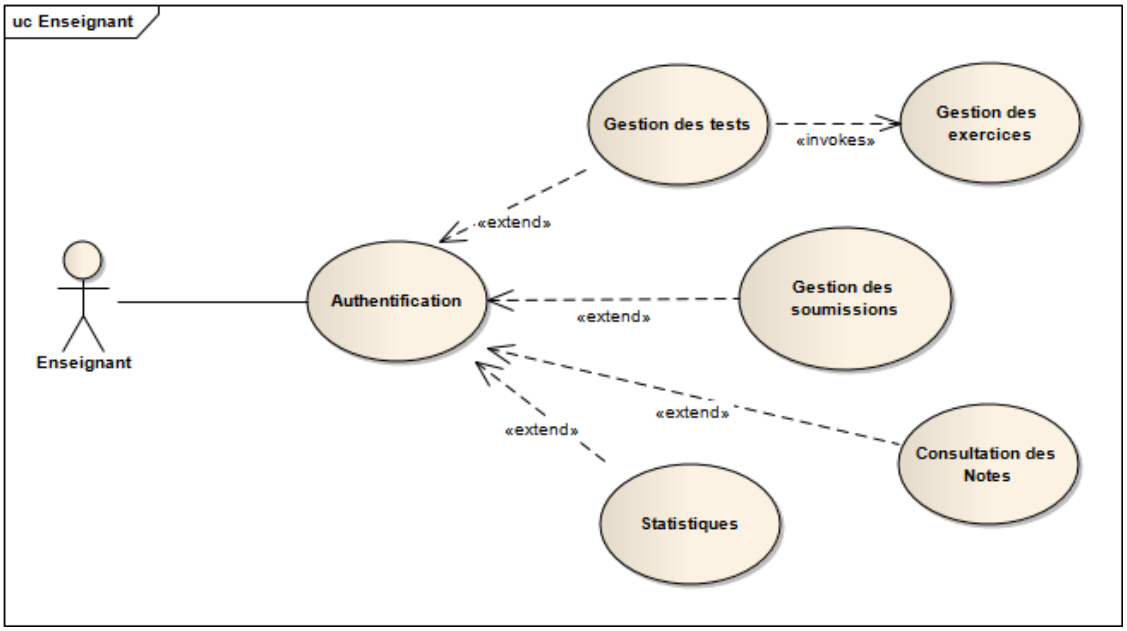

Figure 3: Diagramme de cas d'utilisation de l'enseignant

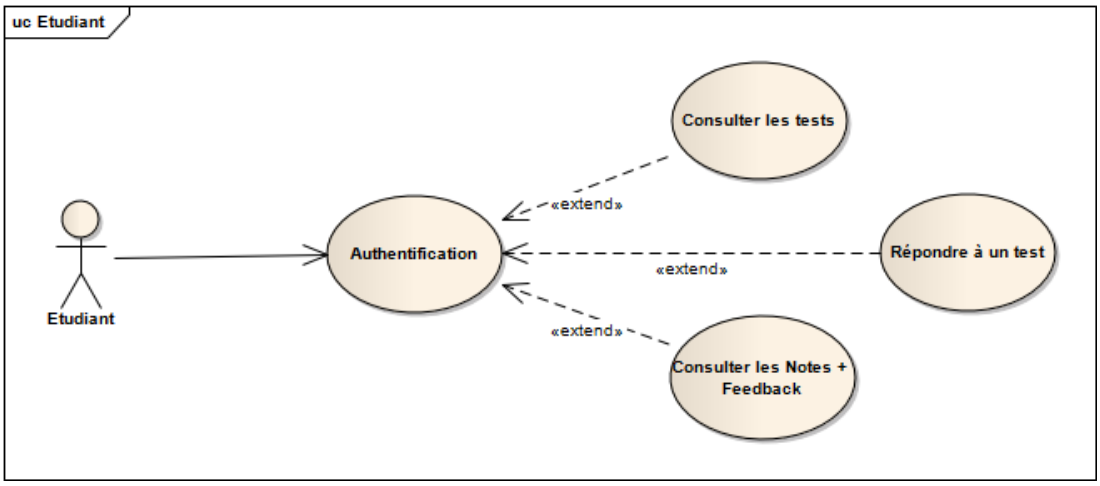

Figure 4: Diagramme de cas d'utilisation de l'étudiant

L'ensemble des composantes de CLAAS sont représentés dans le diagramme de classes suivant. Le diagramme de classes est une représentation statique des éléments qui composent le système et de leurs relations.

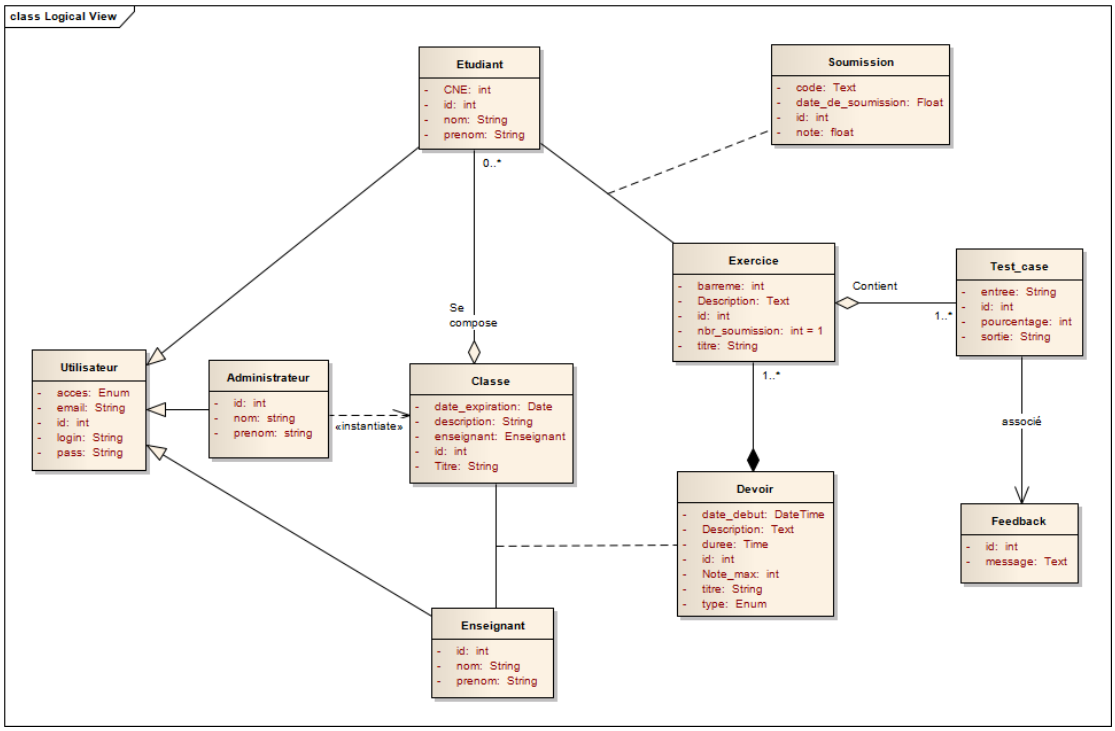

Figure 5: Diagramme de classes de CLAAS

2) Les interfaces de CLAAS

Comme toute application web, l'accès à l'application requiert l'authentification de l'utilisateur. Ainsi, tout utilisateur possède un compte protégé par mot de passe lui permettant d'ouvrir une session sur le système. La création des comptes étant gérée par un administrateur (Fig5).

Figure 6 : Page d'authentification

a. L'interface administrateur

L'administrateur a pour tâche l'alimentation de la base de données du système, en ajoutant les classes, les enseignants et les étudiants.

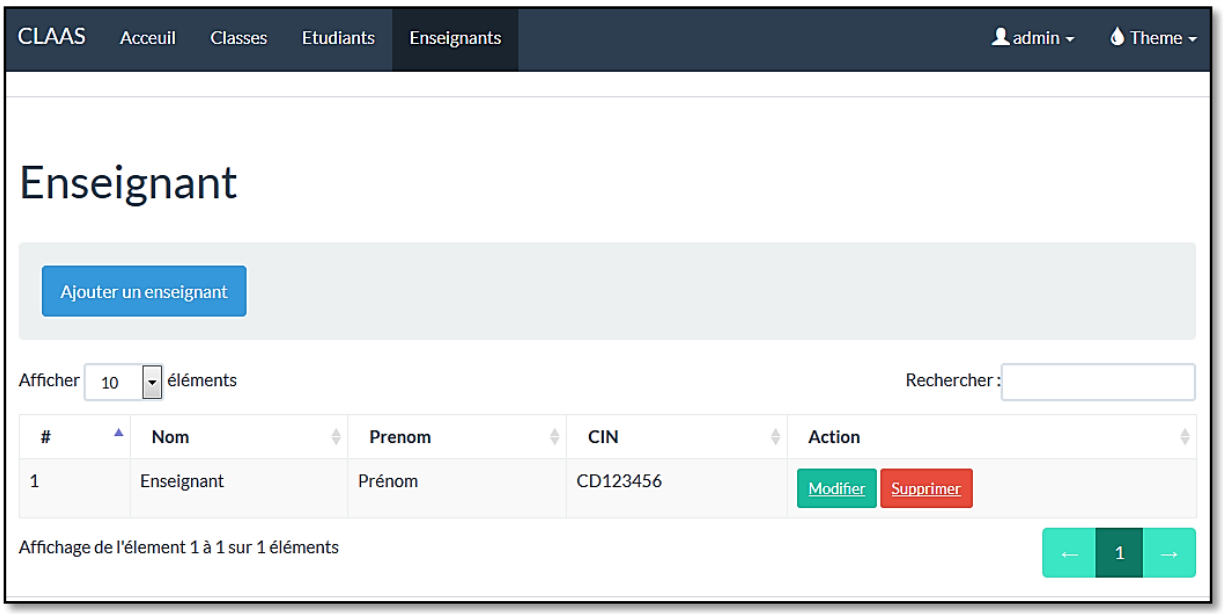

Figure 7 : Compte administrateur - Gestion des enseignants

L'administrateur est responsable de la gestion des classes. La création d'une nouvelle classe consiste à définir un nom pour cette classe, le nom de l'enseignant responsable, la date de fin et un descriptif de la classe. Après la création d'une classe, l'administrateur se charge de déterminer les étudiants affectés à celle-ci, les étudiants déjà ajoutés dans le système. L'enseignant responsable de la classe soit être également ajouté par l'administrateur auparavant.

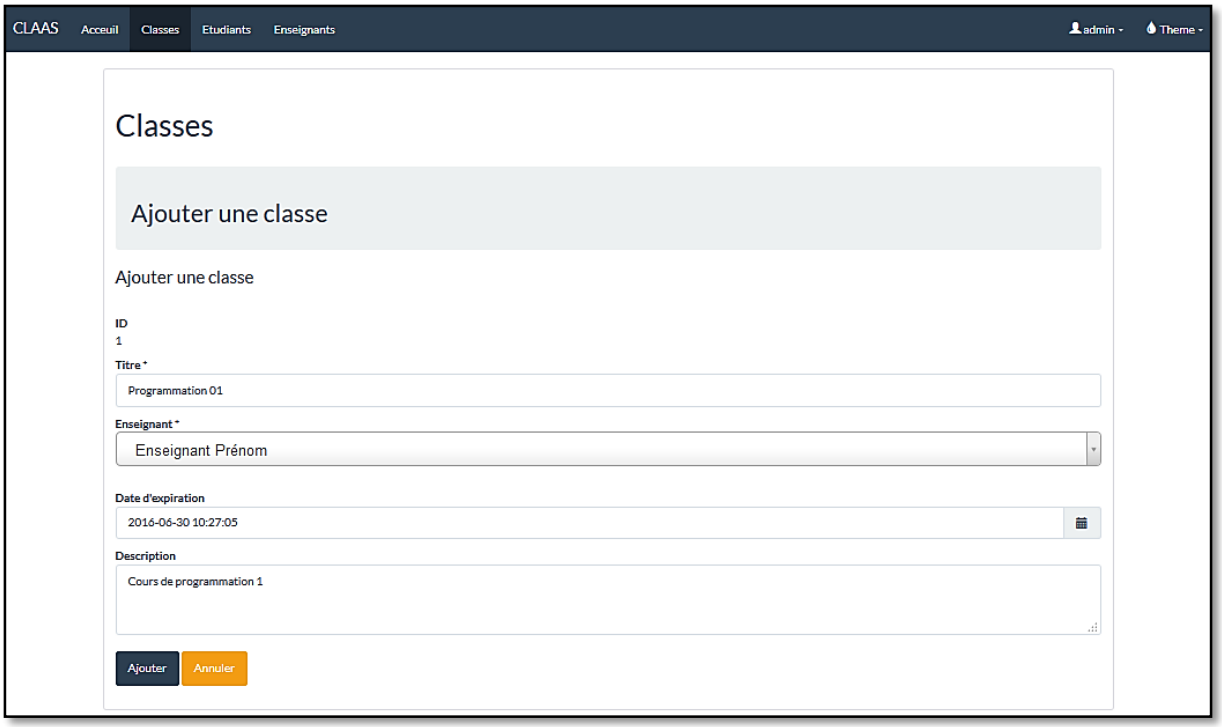

Figure 8 : Compte administrateur - Ajout d'une classe

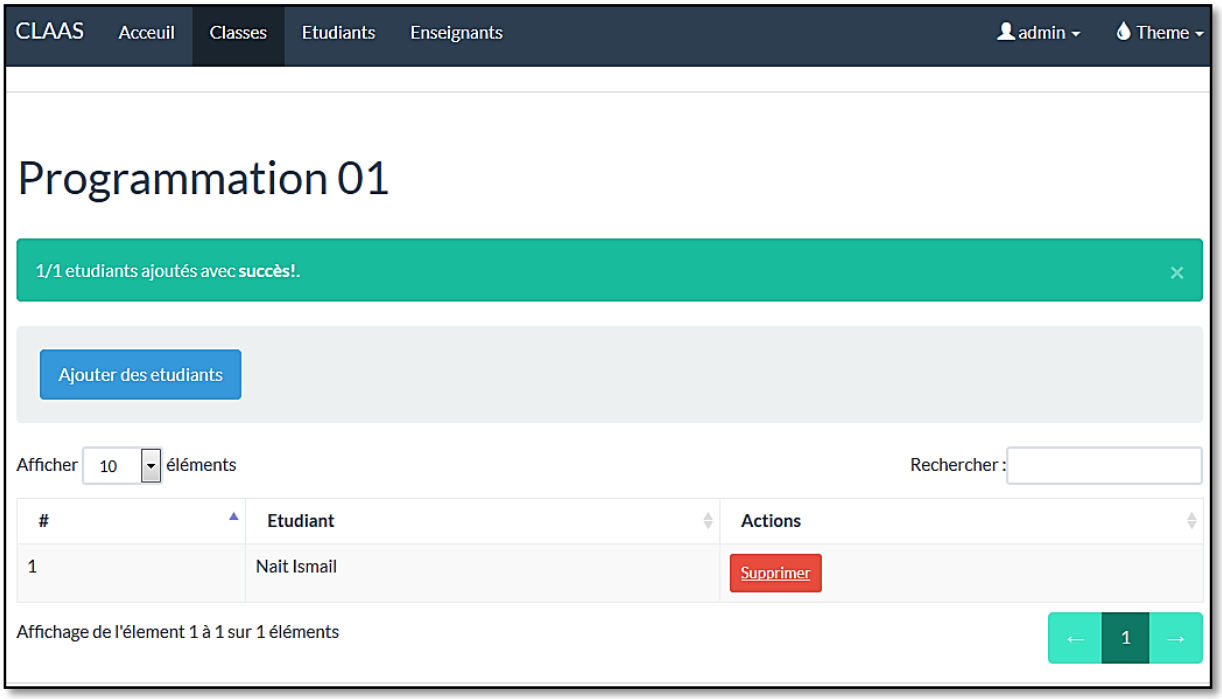

Figure 9 : Compte administrateur - Ajout des étudiants dans une classe

b. L'interface enseignant

L'enseignant peut consulter les classes qu'il enseigne. Avec la possibilité d'exporter toutes les soumissions d'une classe.

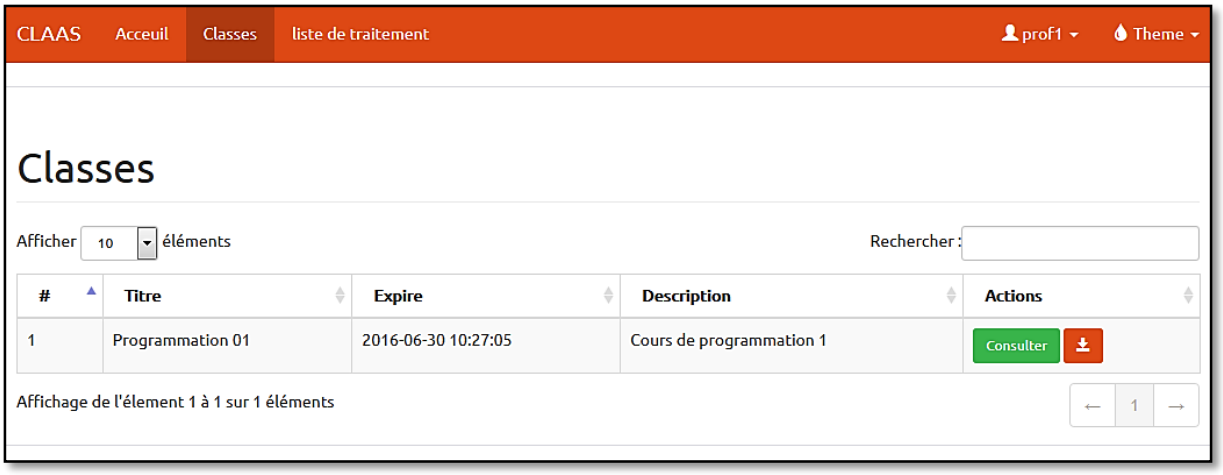

Figure 10 : Compte enseignant - Gestion des classes

La création des épreuves au sein d'une classe existante est une tache de l'enseignant. Celui-ci doit préciser le type de l'épreuve : examen blanc, examen final ou devoir, le titre, la description, la date début, la note maximale et la durée de l'épreuve. Ce dernier champ indique le temps durant lequel l'étudiant peut faire des soumissions.

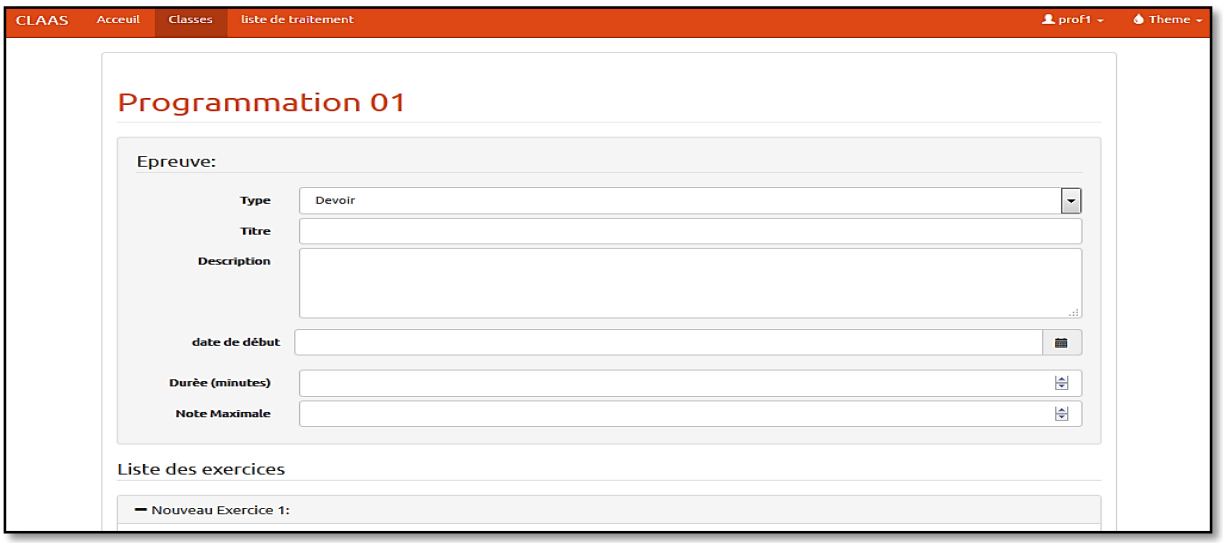

Figure 11 : Compte enseignant - Ajout d'une épreuve

Ensuite, l'enseignant procède à la définition des exercices. L'exercice consiste en un ensemble de composantes : le titre, la description, le barème et les cas de tests correspondants. A noter que le système vérifie si la somme des barèmes affectés aux différents exercices ne dépasse pas la note maximale.

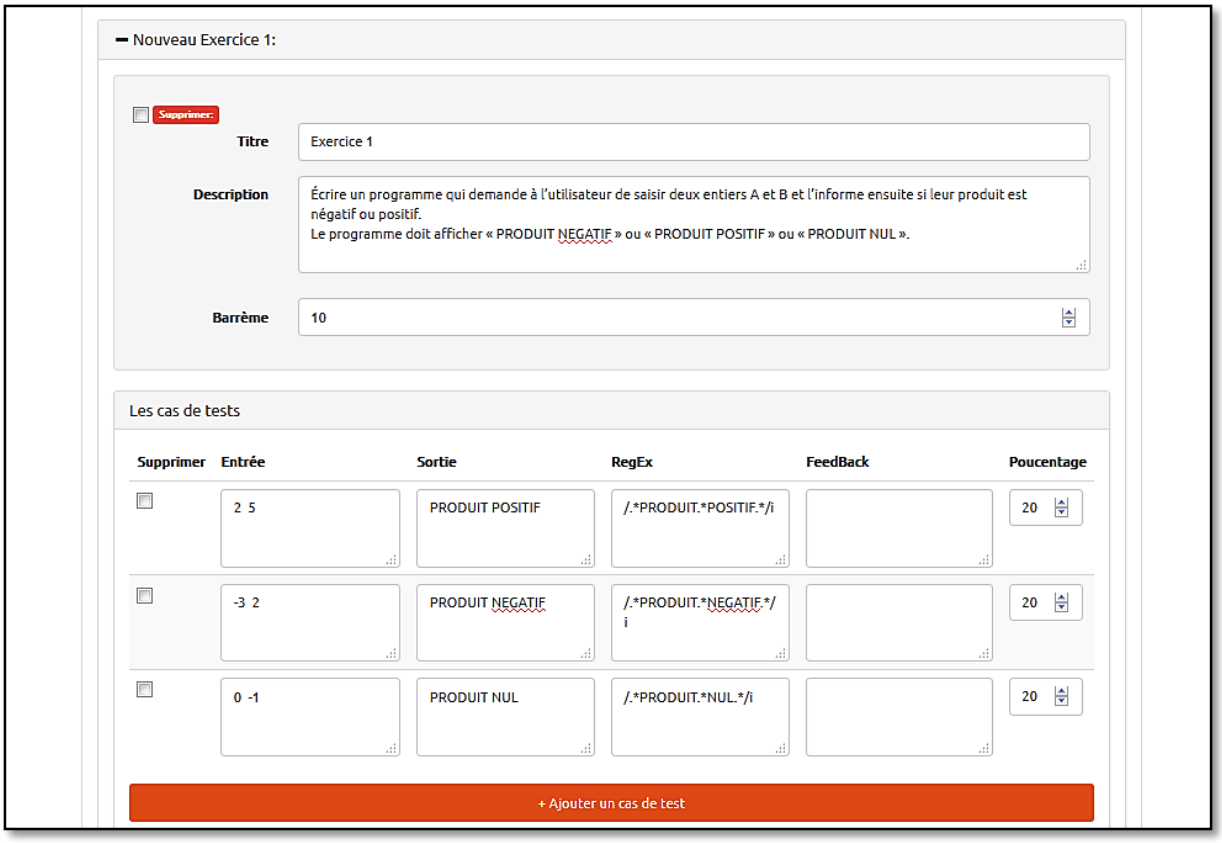

Figure 12 : Compte enseignant – Définition des cas de test

L'ajout d'un exercice requiert la précision d'un barème pour l'exercice ainsi que la définition des cas de test.

La définition des cas de tests est une composante essentielle dans tout système d'évaluation automatique de programme. Il existe plusieurs approches dans ce domaine qui ont tendance à optimiser le choix des données afin de garantir une couverture maximale des chemins d'exécution du programme évalué. Le choix des test-cases influence considérablement la qualité de l'évaluation. Pour le moment, le système proposé se base sur les cas de tests définis par l'évaluateur.

Un cas de test se compose de l'entrée, la sortie attendue et le pourcentage. Eventuellement l'évaluateur peut spécifier une expression régulière qui sera appliqué lors la vérification du format de la sortie.

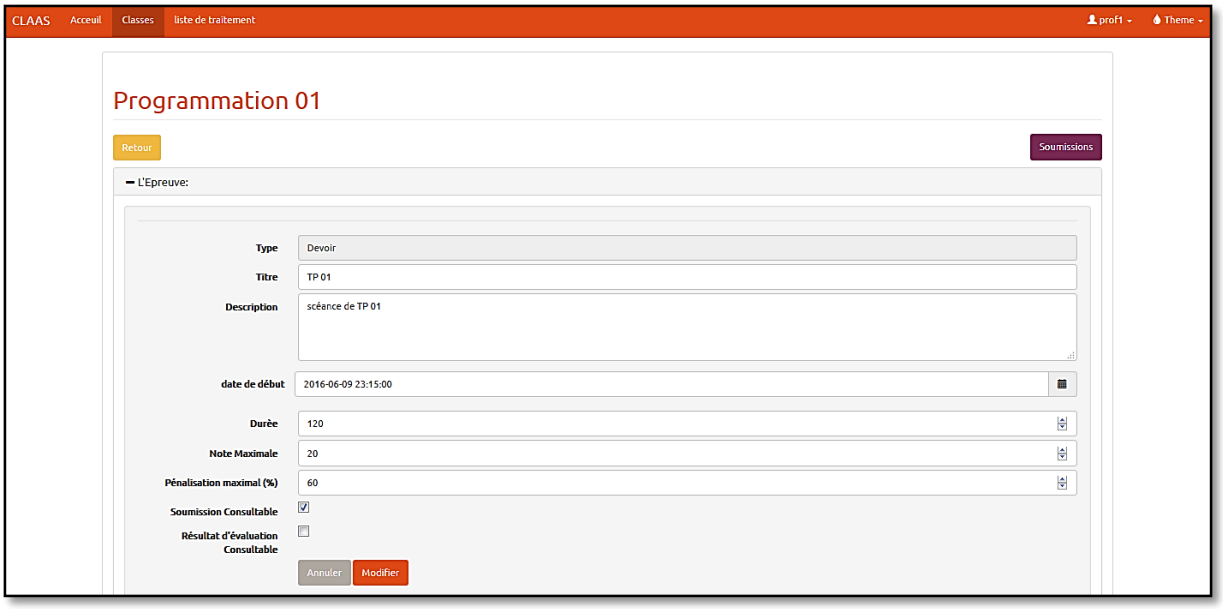

Figure 13 : Compte enseignant - Modification d'une épreuve

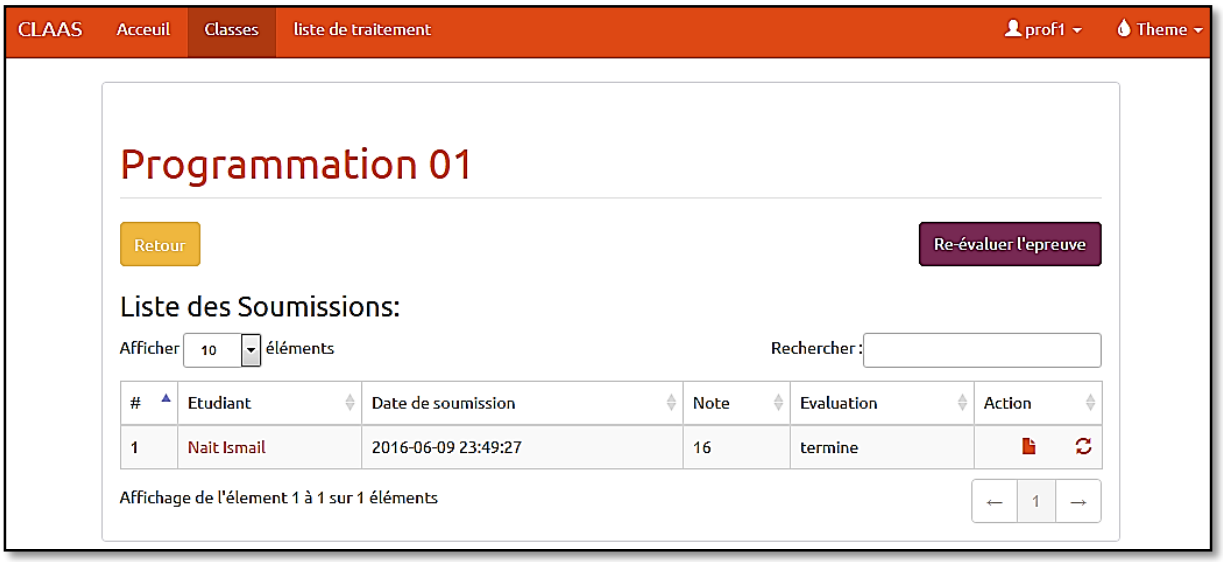

Figure 14 : Compte enseignant - Gestion des soumissions

Le professeur peut visualiser la liste des soumissions d'une épreuve avec la possibilité de réévaluer une soumission ou toutes les soumissions si nécessaire.

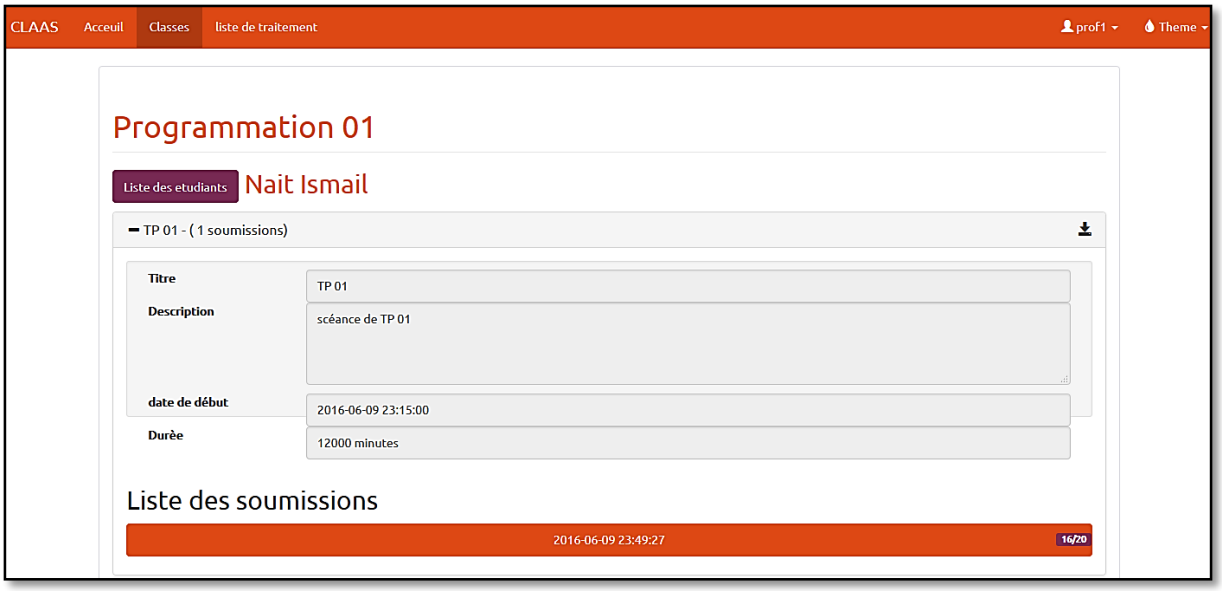

Figure 15 : Compte enseignant - Résultats d'évaluations d'un étudiant

L'enseignant peut également voir toutes les soumissions d'un étudiant particulier avec la possibilité de les exporter.

c. L'interface étudiant :

Après son authentification, l'étudiant accède aux classes auxquelles il a été ajouté.

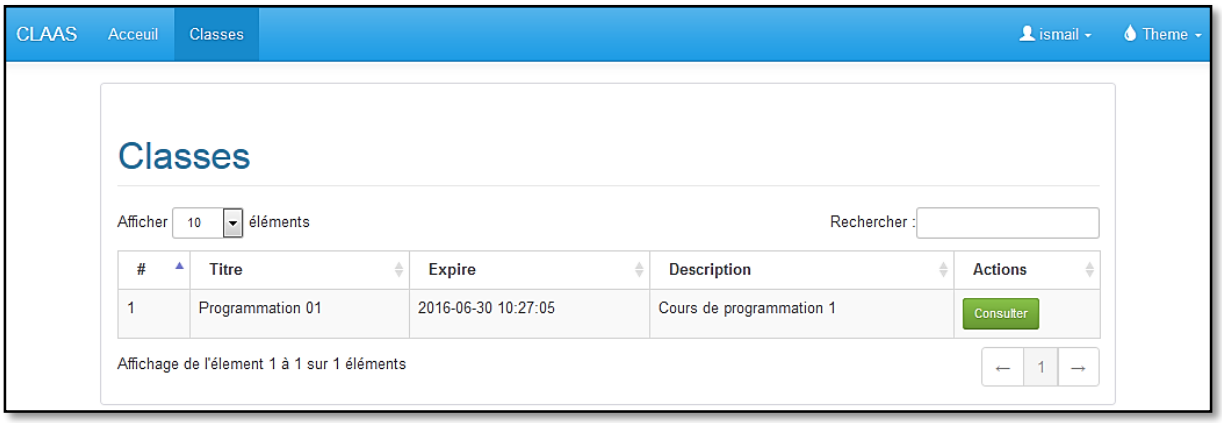

Figure 16 : Compte étudiant - Consultation des classes

Ensuite, l'étudiant peut consulter les épreuves définies dans le cadre de chaque classe.

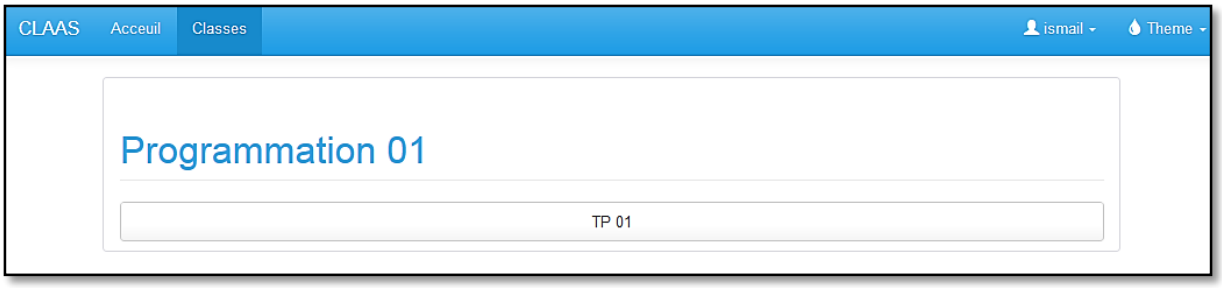

Figure 17: Compte étudiant - Consultation des épreuves dans une classe

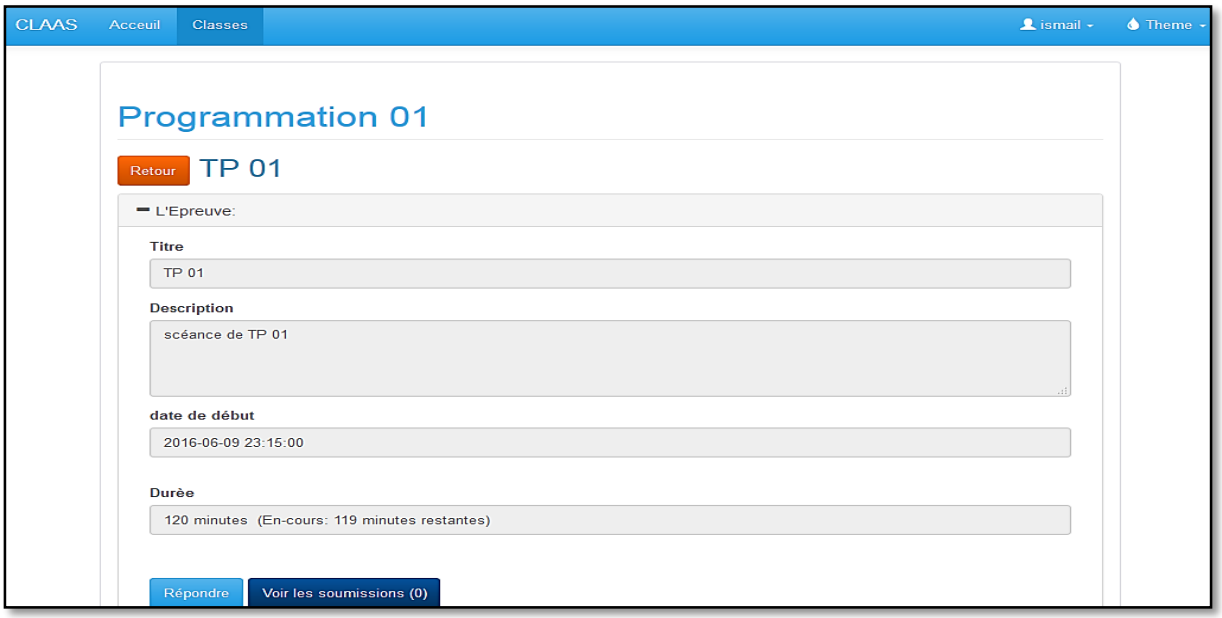

Figure 18 : Compte étudiant - Consultation d'une épreuve

Lors de la consultation d'une épreuve, l'étudiant peut soit répondre à l'épreuve (si elle est en cours) ou consulter ses soumissions. (La possibilité de voir une soumission et/ou le résultat de l'évaluation est contrôlée par l'enseignant).

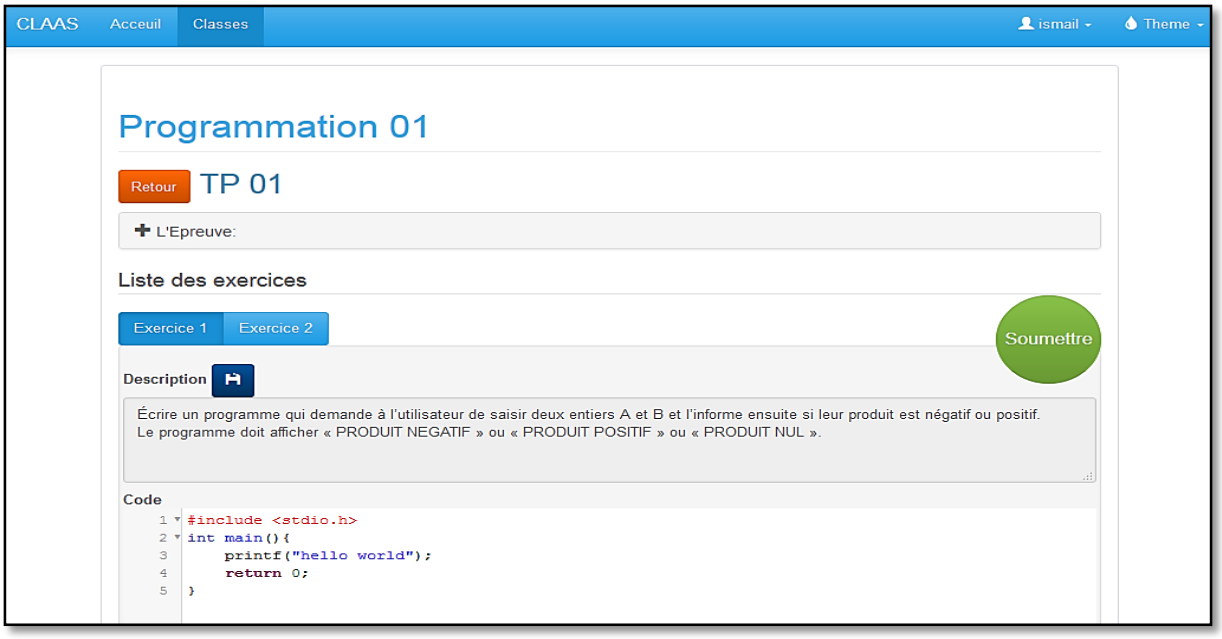

Figure 19 : Compte étudiant - Réponse à une épreuve

Un éditeur est intégré dans CLAAS qui permet à l'étudiant d'éditer son programme. La compilation en ligne (coté serveur) est effectuée à partir de l'éditeur. Le code est enregistrable manuellement (automatiquement lors d'une compilation). Un compteur permet de visualiser le temps restant de l'épreuve.

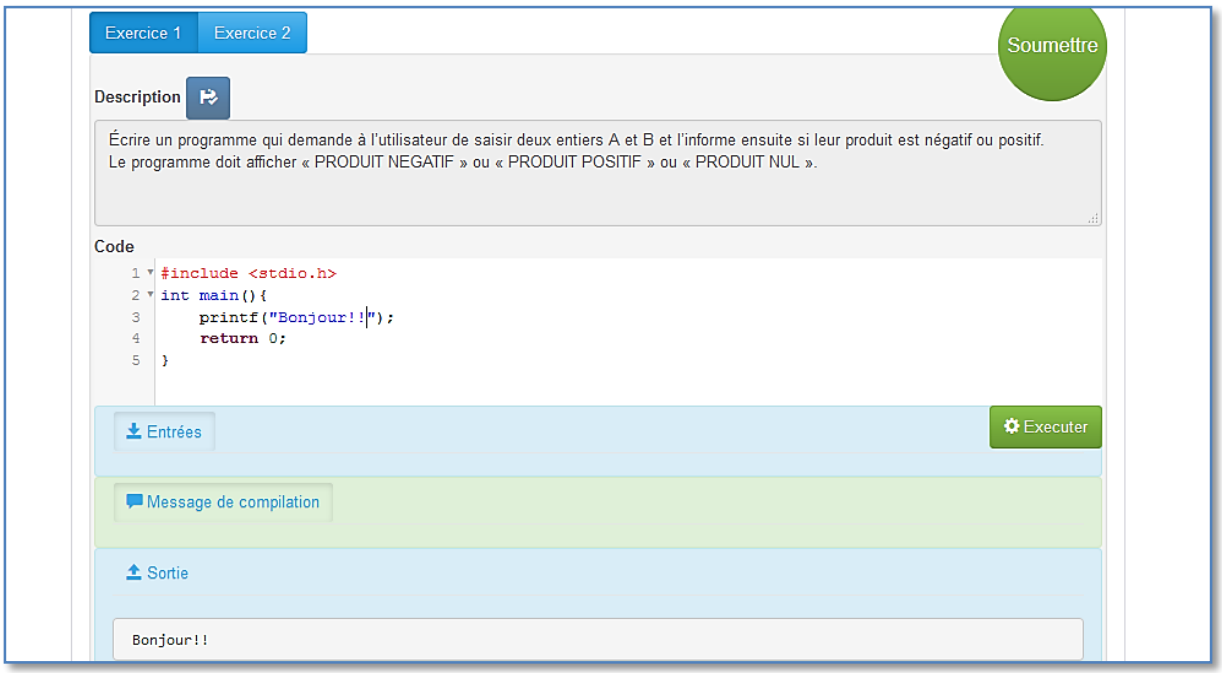

Figure 20 : Compte étudiant - Compilation en ligne du code

L'interface de l'éditeur offre trois volets : la saisie des entrées, l'affichage du message de compilation et l'affichage des sorties du programme. L'exécution du code se fait par clique

sur le bouton « Exécuter » ou par le raccourcie « Ctrl+Entr ». Avant la soumission des programmes, l'étudiant doit vérifier l'ensemble des réponses.

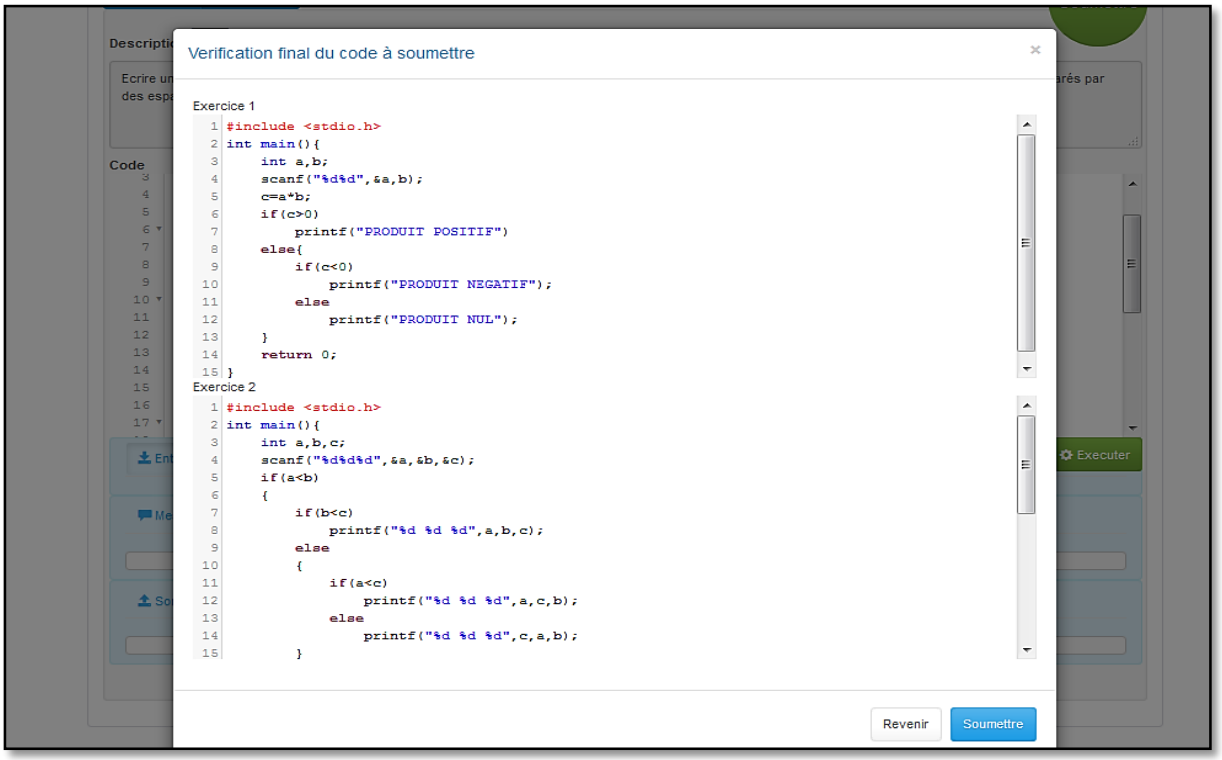

Figure 21 : Compte étudiant - Soumission d'une épreuve

II. Les atouts de CLAAS

A travers la documentation bibliographique effectuée à la base de ce projet, plusieurs manques ont été décelés. Ces manques ou points faibles des systèmes revus ont été exploités de manière à les transformer en atouts dans le système implémenté. Les limites recensées sont majoritairement relatives à la définition des cas de tests. Rappelons que les cas de tests constituent une composante primordiale dans tout système d'évaluation automatique de programme. La qualité du processus de création des tests-cases est nettement rapportée dans le résultat final de l'évaluation automatique d'un programme.

Ainsi, deux améliorations ont été portées à la création des cas de tests par rapport aux systèmes existants sur le marché dans ce domaine. La première amélioration concerne l'affectation d'un pourcentage pour chaque cas de test, permettant ainsi la personnalisation du calcul de la note attribué à l'exercice proportionnellement aux pourcentages fixés. Le pourcentage affecté à un test-case exprime l'importance attribuée par l'évaluateur à ce dernier. A noter que le système vérifie si la somme des pourcentages attribués ne dépasse pas 100%.

La deuxième amélioration concerne la comparaison des sorties de l'exécution du test-case aux sorties attendues. Dans la plupart des systèmes recensés, le contenu des sorties de l'exécution sont précisées dans l'énoncé de l'exercice. Si le fichier contenant les sorties de

l'exécution n'est pas exactement identique à celui contenant les sorties attendues, le programme est alors jugé incorrect.

Afin d'assurer une marge de liberté plus importante dans l'affichage des résultats, les expressions régulières peuvent être utilisées, si l'évaluateur le souhaite, au lieu de donner la description exacte des sorties, chose qui pourrait dévoiler la solution ou une partie de la solution.

A travers l'utilisation des expressions régulières, l'évaluateur fournit les motifs généraux que les sorties du programme évalué doivent respecter au lieu de spécifier les sorties exactes.

III. Expérimentation de CLAAS

L'expérimentation est une étape essentielle qui fait partie du processus de conception et de réalisation d'une application. Dans le but de vérifier la validité et l'efficacité du système réalisé, le système a fait l'objet d'une expérimentation auprès de deux classes de BCG regroupant 37 élèves au total au cours d'une séance de travaux pratiques en langage de programmation C.

L'utilisation d'un système d'évaluation automatique de programme a constitué une expérience inédite pour les étudiants de notre université. Afin de prévoir les tendances d'utilisation du système réalisé par les étudiants dans l'avenir, un questionnaire a été utilisé dans le but de collecter les données nécessaires à cette fin.

Ce questionnaire regroupe 7 questions relatives à la facilité d'utilisation du système, le besoin d'assistance durant son utilisation, l'expérience avec d'autres systèmes d'évaluation automatique de programmes, la qualité de l'interface, la valeur ajoutée par rapport à la méthode traditionnelle, l'adoption de ce système dans les examens. Le questionnaire est présenté en annexe à ce mémoire.

A travers l'analyse des informations collectées à travers les réponses des étudiants à ce questionnaire, on a pu relever la satisfaction exprimée par ceux-ci pour l'utilisation d'un outil d'évaluation automatique de programmes. Le tableau suivant résume les résultats du questionnaire :

Questions du questionnaire	Réactions des étudiants
La facilité d'utilisation du système	Facile à très facile
Le besoin d'une assistance humaine	No
La qualité de l'interface utilisateur	Moyenne à bonne
S'ils préfèrent l'utilisation d'un système automatique au lieu de la méthode traditionnelle durant l'examen	Oui

Tableau 1 : résumé du questionnaire et des réponses des étudiants

L'expérimentation du système réalisé regroupe les cinq exercices suivants :

- ***Exercice1*** : Ecrire un programme qui demande à l'utilisateur d'entrer un nombre entier et affiche si le nombre est positif ou négatif.
- ***Exercice2*** : Ecrire un programme qui lit 5 notes comprises entre 0 et 20 et indique combien d'entre elles sont supérieures à la moyenne 10.
- ***Exercice3*** : Ecrire un programme qui demande à l'utilisateur d'entrer deux nombres réels puis lui propose le menu suivant :

  1 : Somme

  2 : Différence

  3 : Produit

  4 : Quotient

  Le résultat de l'opération choisie est affiché à l'écran.
- ***Exercice 4*** : Ecrire un programme qui demande à l'utilisateur de saisir un nombre entier au clavier et affiche une colonne contenant un nombre d'étoiles correspondant au nombre donné, par exemple :

  Saisir un nombre entier :

  3
  *
  *
  *
- ***Exercice 5*** : Ecrire un programme qui demande à l'utilisateur de lui fournir un nombre entier positif et inferieur a 100 et ceci jusqu'à ce que la réponse soit satisfaisante. Le dialogue se présentera ainsi :

  Donnez un entier positif inférieur à 100 : 452
  SVP un entier positif inférieur à 100 : 0
  SVP un entier positif inférieur à 100 : 28
  Réponse correcte pour le nombre 28

Le choix des exercices est fait par rapport à l'objectif ciblé par cette séance. L'objectif de cette séance de travaux pratiques étant d'évaluer les compétences des étudiants à utiliser les différentes structures répétitives et itératives.

Durant l'expérimentation, un barème est affecté à l'épreuve. Egalement, pour chaque exercice, différents tests-cases sont fournis. Pour chaque test-case, on attribue un poids sous forme d'un pourcentage pris en compte lors du calcul de la note attribuée au programme évalué.

Prenons comme exemple l'exercice 4 qui affiche des étoiles en fonction du nombre entré par l'utilisateur (voir Fig.12)

A travers la figure 12, on remarque de le pourcentage attribué aux derniers test-case est le plus grand, c'est-à-dire une importance plus grande est attribuée au fait que l'étudiant doit tester, à travers le programme soumis, si la donnée entrée n'est pas un nombre.

Ainsi, si le programme évalué omet ce test, une réduction de 30% est appliquée à la note attribuée à cet exercice.

Afin de mesurer la pertinence du système réalisé, on a comparé les résultats de l'évaluation des différents programmes effectuée par celui-ci aux résultats de l'évaluation humaine ou manuelle. Les graphiques suivants représentent les résultats de cette comparaison.

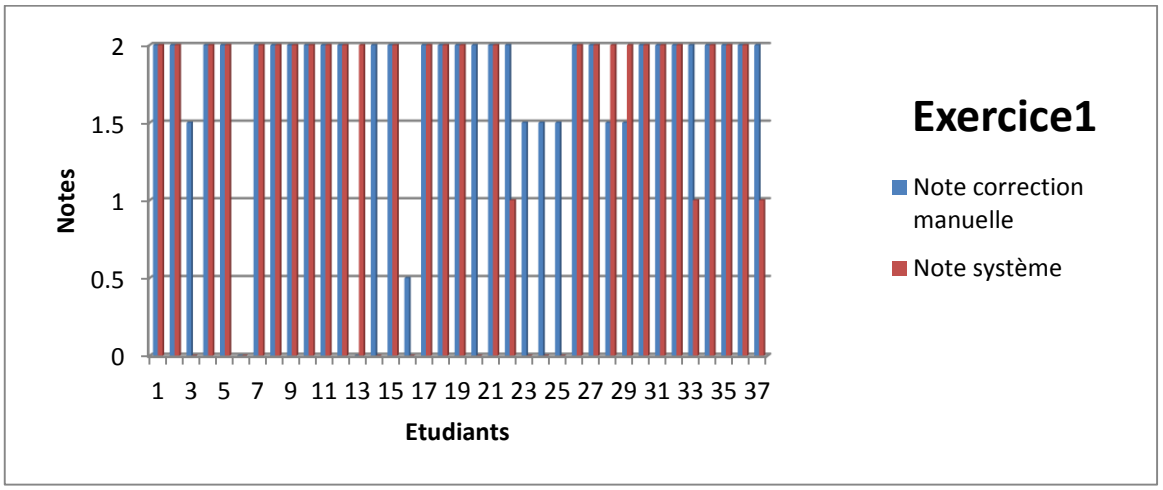

Figure 22: Notes de correction de l'exercice1

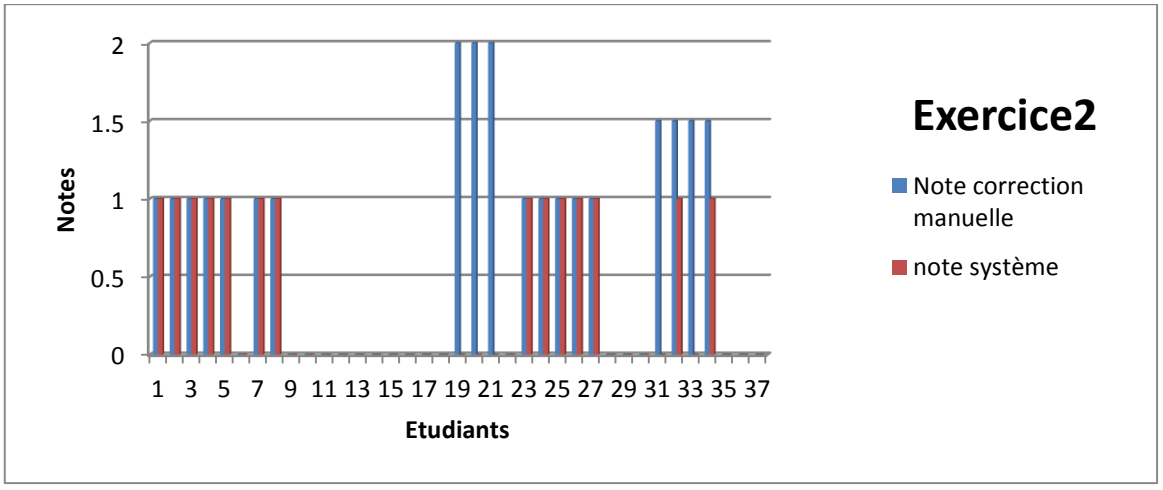

Figure 23: Notes de correction de l'exercice2

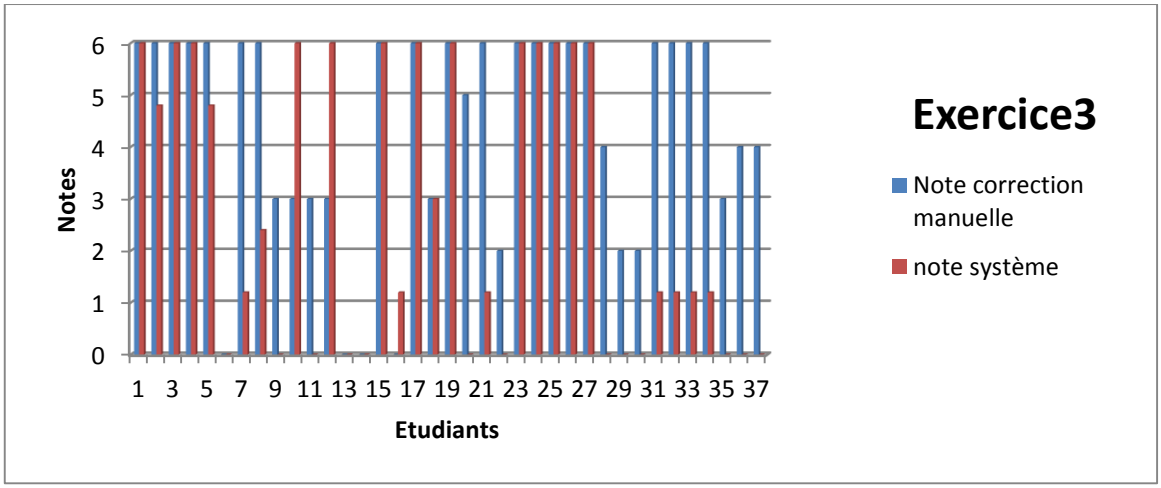

Figure 24 : Notes de correction de l'exercice 3

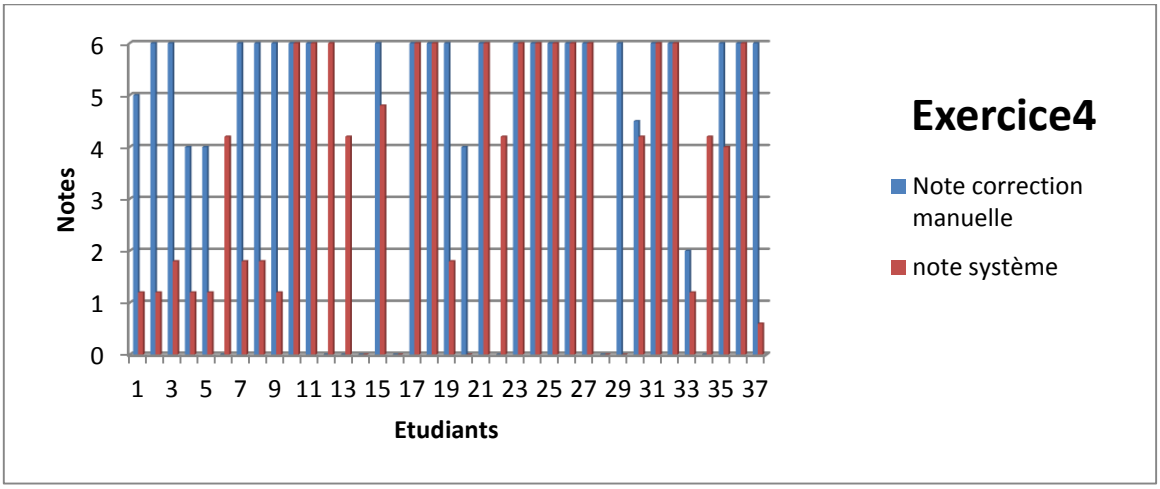

Figure 25: Notes de la correction de l'exercice 4

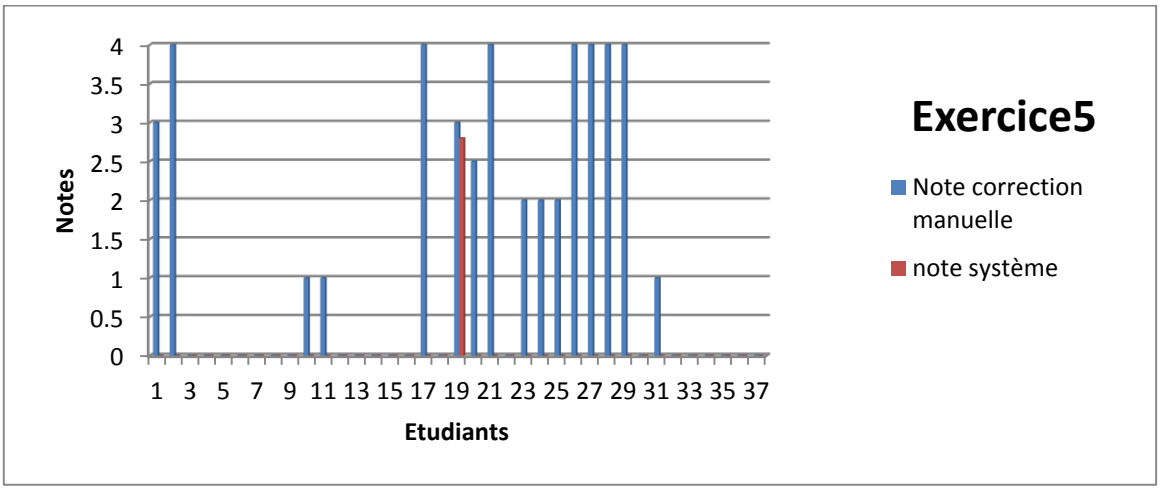

Figure 26 : Note de la correction de l'exercice5

A travers les graphes ci-dessus, on remarque que l'évaluation par CLAAS donne globalement des résultats identiques ou très rapprochés des résultats de l'évaluation humaine. Cependant, on remarque clairement que dans certains cas, la note de la correction automatique est considérablement différente de celle de la notation manuelle. Cela est expliqué ainsi :

- Dans le cas où la note fournie par le système excède la note de la correction manuelle : suite à la revue des programmes en question, on a constaté que certains programmes réussissent les cas de tests et donc sont considérés corrects de point de vue de l'évaluation automatique effectuée par CLAAS. Cependant, ces programmes-là, même s'ils sont corrects, ils ne répondent pas aux exigences de l'évaluateur, par conséquent, l'évaluateur réduit la note attribuée, ce qui explique la différence relevée. Par exemple, dans le cas du troisième exercice, l'évaluation manuelle prend en considération l'utilisation de « Switch » dans les programmes des étudiants. Toutefois, on a trouvé que certains programmes répondent au problème en utilisant une suite de « IF..Else ». Puisque la méthode d'analyse dynamique ne s'intéresse pas au code-source du programme évalué, on ne peut vérifier l'utilisation d'une méthode précise dans le programme, de ce fait, la justesse de celui-ci dépend uniquement de la réussite des tests-cases.
- Inversement à ce qui a été discuté dans le paragraphe précédent, dans le cas où le programme évalué ne se compile pas, alors le processus de son évaluation par CLAAS est interrompu. Cependant, l'évaluateur humain peut quand même attribuer une note à ce programme tout en pénalisant les erreurs qu'il contient. Dans ce cas, la note fournis par CLAAS est nulle, tandis que le programme est noté par l'évaluateur humain.

Figure 25 présente la tendance générale des deux courbes représentant la moyenne des notes des étudiants pour chaque exercice selon les résultats produits par CLAAS et par l'évaluation manuelle.

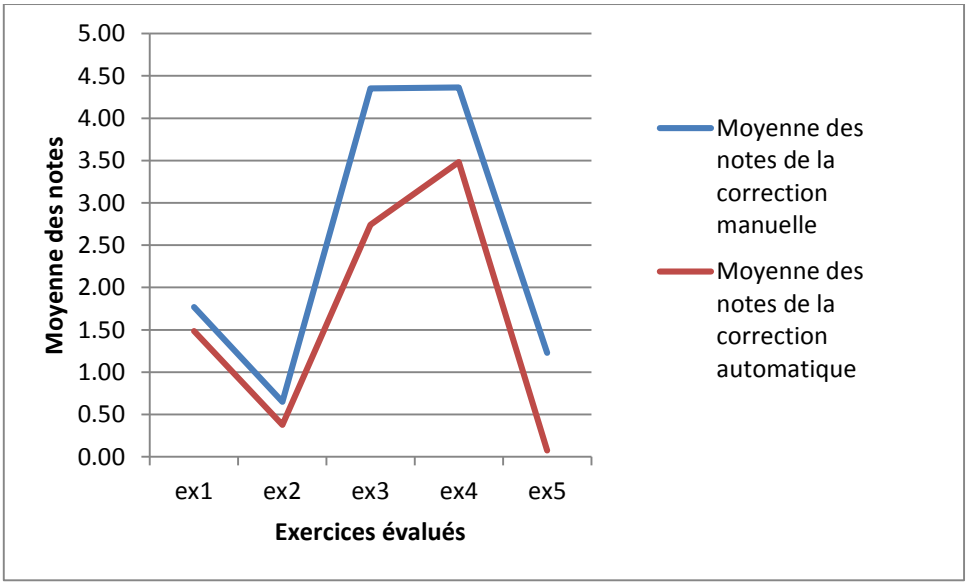

Figure 27 : Tendance globale des notes de l'évaluation automatique et manuelle

A travers cette figure, on peut voir que la courbes représentant les résultats de la notation manuelle et celle représentant les résultats de l'évaluation effectuée par CLAAS varient parallèlement tout en ayant la même tendance.

IV. Points faibles et limites

A travers l'utilisation du système CLAAS, nous avons constaté quelques limites. Ces limites sont dues à la méthode sur laquelle CLAAS se base pour l'évaluation des programmes, à savoir la méthode d'analyse dynamique.

L'analyse dynamique de programme consiste à exécuter le programme en utilisant les données des cas de tests suggérés. Par conséquent, l'évaluation d'un programme en utilisant la méthode d'analyse dynamique est conditionnée par son exécution, et nécessairement par sa compilation réussie.

Dans le cas où un programme contient des erreurs de compilation et/ou d'exécution, le processus d'analyse est interrompu et le programme ne peut être évalué. Ainsi la note attribuée par CLAAS dans ce cas est 0. Cela peut s'avérer décourageant pour les étudiants qui ne reçoivent comme feed-back que les messages d'erreurs générés suite à l'échec de la compilation ou de l'exécution de leurs programmes. Cela implique que l'étudiant doit chercher lui-même l'origine de l'erreur commise, ce qui peut s'avérer difficile dans le cas des étudiants novices en programmation.

Cette limite est présente dans tous les systèmes basés sur la méthode d'analyse dynamique. Dans CLAAS, on a essayé de proposer une solution afin de contourner cette limite. Le chapitre III décrit la solution innovante proposée en vue de porter une amélioration de la méthode d'analyse dynamique.

Chapitre3. Intégration de la détection, localisation et correction automatique d'erreurs dans CLAAS

"Si le débogage est le processus d'élimination des bugs, alors la programmation doit être le processus de les mettre dedans." – Edsger Dijstra.

I. Présentation et description

Nous rappelons brièvement que la méthode d'analyse dynamique présente des avantages multiples mais souffre aussi d'inconvénients contraignants. Parmi les limites de cette méthode on cite l'incapacité d'analyser un programme contenant des erreurs qui causent l'échec de sa compilation et de son exécution, puisque la dite méthode se base sur les sorties de l'exécution du programme évalué.

Cette limite ne pose pas de problème dans le cas de l'évaluation de programmes soumis par des expérimentés dans le domaine de programmation lorsqu'elle est utilisée, à titre d'exemple, par les systèmes de gestion de compétitions en programmation.

Néanmoins, dans le cadre de l'évaluation de programmes soumis par des étudiants novices dans le cours de programmation, l'utilisation de la méthode d'analyse dynamique avec sa conception originelle s'avère inappropriée. Lorsqu'un programme est évalué en se basant sur la méthode d'analyse dynamique, s'il contient des erreurs de compilation ou d'exécution, il est jugé incorrect même quand l'algorithme sur lequel il a été fondé est correct ou partiellement correct. De ce fait, le processus de l'évaluation du programme est interrompu à ce stade. Elle ne peut être achevée que si le programme réussit la phase de compilation et d'exécution.

Selon l'approche proposée dans le cadre de ce travail, cette limite majeure peut être surmontée en procédant à améliorer la méthode d'analyse dynamique. Cette amélioration est proposée à travers l'automatisation de la localisation et de la correction des erreurs détectées. Ceci augmentera les chances pour un programme d'être évalué, et évitera ainsi le découragement chez l'étudiant causé par l'interruption du processus d'évaluation et son incapacité parfois à détecter ou à corriger les erreurs commises.

En adressant cette question, nous traitons implicitement une autre limite de la méthode d'analyse dynamique à savoir l'insuffisance remarquée relativement au feed-back délivré suite à l'évaluation d'un programme. Selon la conception native de la méthode d'analyse, le feed-back desservis n'est autre que le contenu des logs de compilation dans le cas de l'échec du programme à se compiler, dans le cas échéant, une liste des sorties des tests-cases échoués est retournée.

II. Catégorisation des erreurs

Avant d'attaquer le problème de la correction des erreurs dans les programmes. Il faudrait tout d'abord proposer une classification de ces erreurs, ceci afin de simplifier le processus d'évaluation que ça soit pour la pénalisation ou pour les feedbacks générés.

De façon générale, les erreurs dans un programme peuvent être classifiées selon le stade de vie du programme durant lequel l'erreur apparaît, à savoir les stades de compilation, d'édition de liens ou d'exécution. Cette classification pour se résumer comme suit :

1) Erreurs de compilation :

Les erreurs de compilation, également connues sous le nom d'erreurs du compilateur, sont des erreurs qui empêchent un programme de s'exécuter.

Du point de vue d'un compilateur, les erreurs sont classifiées dans trois catégories :

- Erreurs lexicales

  Les erreurs lexicales sont détectées lors de l'analyse syntaxique du code source. Une telle erreur se produit quand un lexème (unité de programme indivisible) n'existe pas dans la liste des symboles du langage.

- Erreurs syntaxiques

  Une erreur de syntaxe est détectée lorsque le *parser* (l'analyseur syntaxique) attend un symbole qui ne correspond pas au symbole en cours.

  Exemples : un point-virgule manquant, variable non déclarée, parenthèses non appariées, etc…

- Erreurs sémantiques

  Une erreur sémantique indique un usage inapproprié d'une instruction.

  Exemples : Utilisation d'une variable non initialisées, types incompatibles, Index de tableau hors de portée, etc…

2) Erreurs d'édition de liens

Une erreur de liaison peut se produire après la compilation. Cela signifie que le code se compile bien, mais une fonction ou une bibliothèque qui est nécessaire ne peut être trouvée.

3) Erreurs d'exécution :

Les erreurs d'exécution se produisent lors de l'exécution d'un programme.

Ce genre de bugs est généralement imperceptible par le compilateur et souvent due à une mauvaise entrée de l'utilisateur.

Exemple : Division par zéro, ouverture d'un fichier inexistant, etc…

4) Erreurs de logique :

En plus des trois catégories précédentes, on peut ajouter la catégorie des erreurs de logique. Une telle erreur empêche le programme de faire son comportement prévu : Le code peut se compiler et s'exécuter sans erreur, mais une opération peut produire un résultat inattendu.

Ce genre d'erreurs est difficile à être détecter et nécessite un examen profond du programme.

Dans le contexte de ce travail, on vise à corriger les erreurs situées parmi les trois premières catégories citées à savoir : les erreurs de compilation, d'édition de liens et d'exécution. A présent, on exclue la dernière catégorie vu la complexité de détecter les erreurs logique.

Nous avons ainsi proposé une classification d'erreurs en se basant sur la base d'erreurs fréquentes recueillies par le système CLAAS durant l'expérimentation présentée dans le paragraphe3 du deuxième chapitre [12].

Catégories d'erreur	Exemples
Fautes d'orthographe	Erreur dans les noms d'une fonctions..
Erreurs de symboles	Un point-virgule manquant ou superflu, & manquant, etc.
Erreurs de typage	Type de variable incompatible..
Erreurs de bibliothèque	#include«stdio.h» manquante, etc..
Erreurs de logique	& à la place de &&, = à la place de ==, etc.
Erreurs de Variable	Variable non declarée ou non initilisée..
Erreurs de boucle	Problème dans la structure d'une boucle..

Figure 28 : Exemples de catégories d'erreurs

III. Unité de correction

Pour corriger une erreur, il faut tout d'abord la détecter puis la localiser. Le but principal de ce travail de recherche est d'intégrer dans CLAAS la fonctionnalité qui permet de détecter et de corriger automatiquement la plupart des erreurs entravant le programme à passer l'étape de compilation et à s'exécuter.

Le compilateur reste un outil puissant pour la détection de la majorité des erreurs. Pour cette raison on a opté à exploiter les messages d'erreurs produits par le compilateur afin de détecter les erreurs commises.

En effet, après la compilation du code, en récupère le message du compilateur. Ce message est analysé dans le but d'extraire les erreurs trouvées. Ces erreurs sont ensuite passées au correcteur pour les corriger et générer le nouveau code.

La figure 14 présente le schéma général de la méthode proposée de l'évaluation automatique des programmes C.

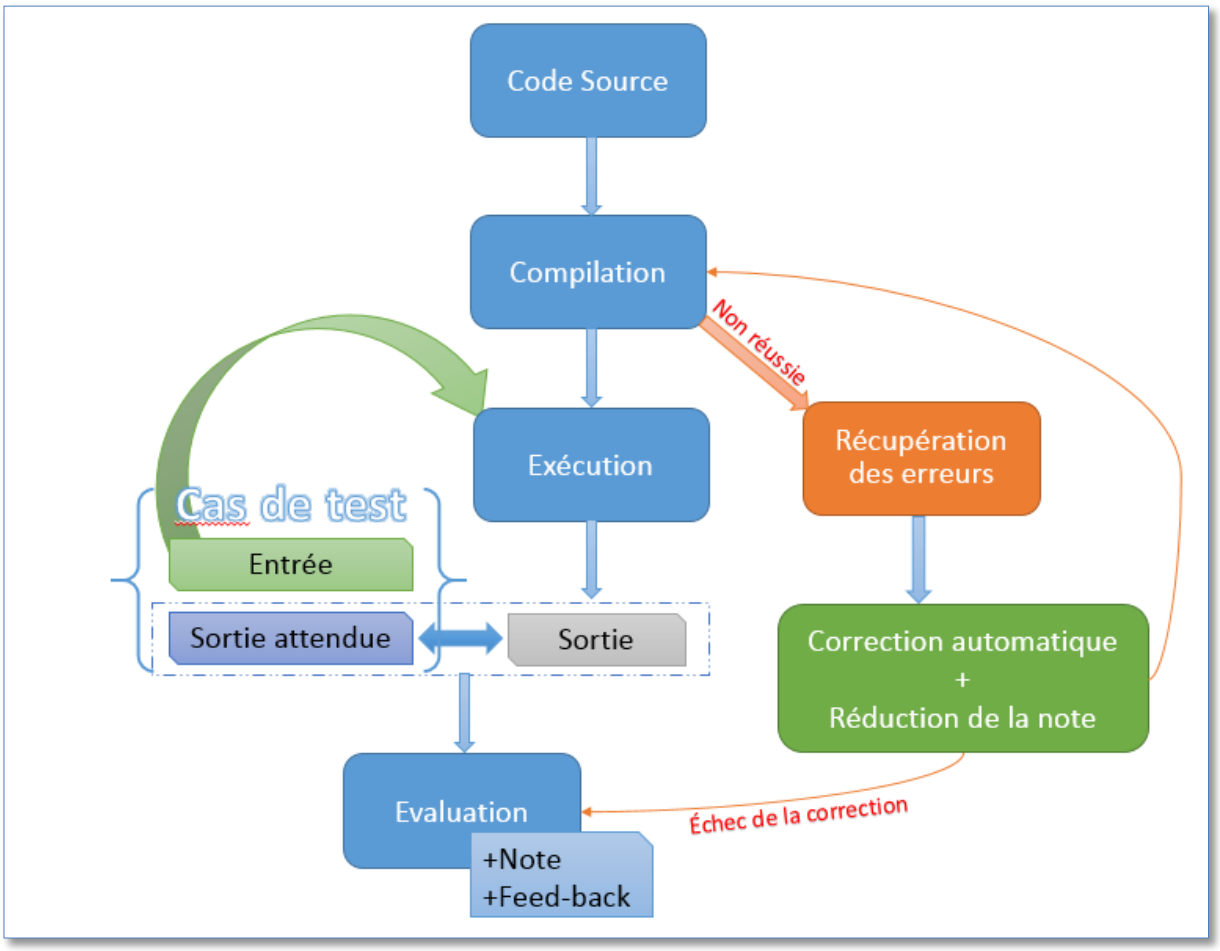

Figure 29: Schéma de l'analyse dynamique avec correction

L'introduction de l'unité de correction dans le processus d'évaluation dynamique a pour but l'amélioration des résultats de l'évaluation. Durant le processus d'évaluation normal, l'existence d'une erreur peux stopper l'évaluation résultant en une note défavorable.

Le processus de correction commence par la détection des erreurs, via l'analyse du message de compilation. Ensuite, on passe à la phase de localisation des erreurs. Quand le compilateur indique l'existence d'une erreur, il indique seulement l'emplacement où l'erreur s'est produite est non pas l'origine de l'erreur. Le rôle de la localisation des erreurs est de trouver l'emplacement exact de l'origine de cette erreur. Ce qui nécessite une analyse syntaxique du code.

Dans notre approche, on traite les erreurs individuellement en proposant une correction adéquate de chaque erreur selon le message d'erreur généré par le compilateur.

Les messages de compilation révèlent l'existence d'erreurs dans le programme évalué. Ces erreurs sont soit localisées par le compilateur, ou doivent par la suite être localisées par l'unité de correction.

Considérant une erreur simple, telle que l'oubli du point-virgule à la fin d'une instruction, la plupart des compilateurs donnent des messages pertinents sur l'emplacement ce genre d'erreurs :

```
code.c:3:23: error: expected ';' after expression
```

Le compilateur a détecté l'erreur et l'a également localisée (3:23). La localisation du compilateur est utilisée comme valeur initial de localisation pour l'unité de correction sur laquelle elle se base pour appliquer la correction nécessaire [13], dans ce cas, l'insertion du point-virgule à l'emplacement adéquat.

Pour résumer, voici le schéma général du processus de la correction :

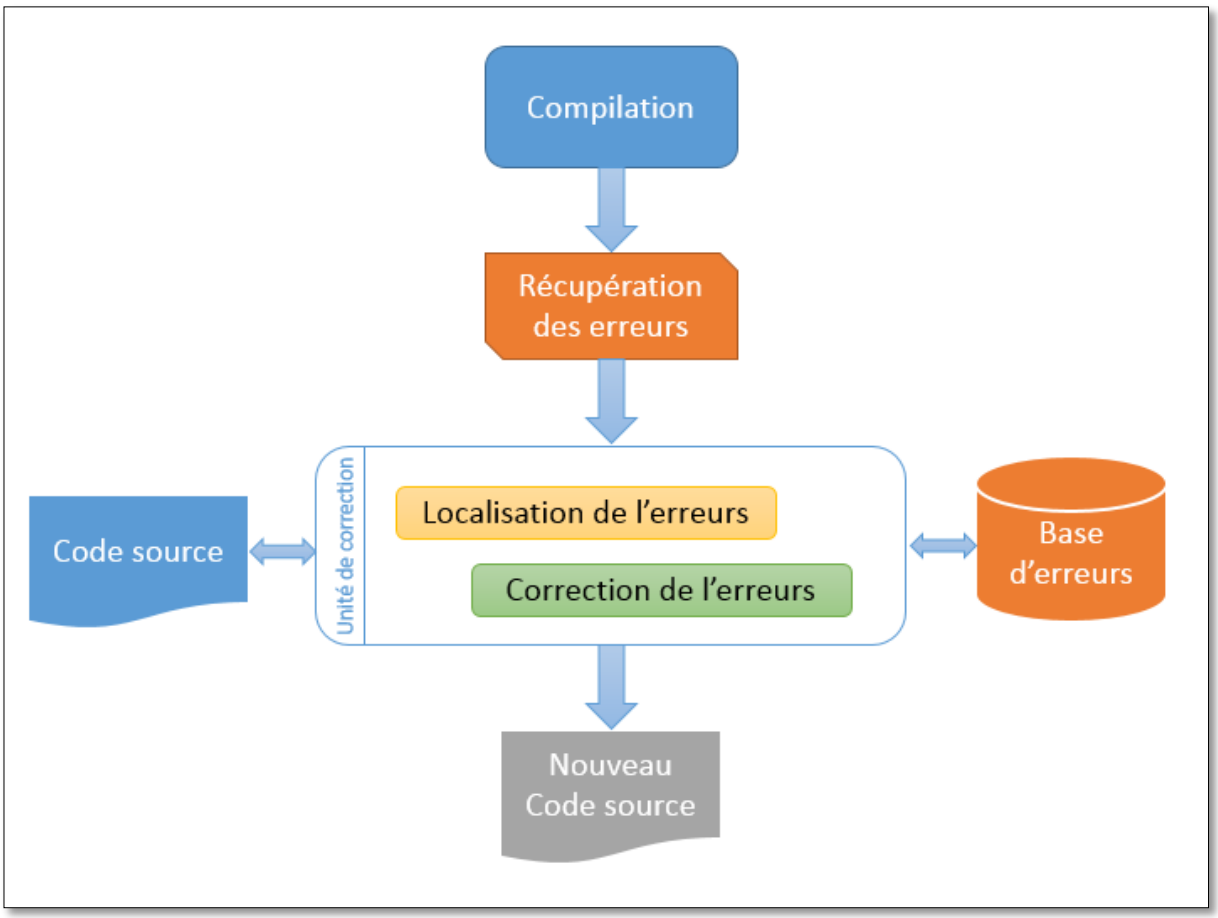

Figure 30 : Processus de correction

IV. Evaluation et pénalisation

Comme il a été noté précédemment, l'objectif principal de ce travail est l'évaluation des programmes. Ce qui consiste à fournir une note et un feed-back à la fin de l'évaluation.

Rappelons que le principe de l'évaluation automatique par la méthode d'analyse dynamique se base sur l'exécution du programme et son évaluation contre un ensemble de cas de test prédéfinis. Quand le code soumis contient des erreurs, son exécution devient impossible et son évaluation via l'analyse dynamique aussi.

Lorsque l'unité de correction réussit à corriger les erreurs contenues dans le programme, une pénalisation sera effectuée suite à chaque correction selon la catégorie des erreurs et leurs occurrences, par la suite, c'est le nouveau code corrigé qui sera exécuté et évalué.

En effet, pour chaque catégorie d'erreurs, l'évaluateur définie un pourcentage de pénalité et associe un feed-back. Ceci est représenté par des paramètres que l'évaluateur fixe dans CLAAS avant de procéder à son utilisation. Ainsi, l'évaluateur est libre de personnaliser l'évaluation selon ses objectifs. L'évaluateur peut éventuellement définir un seuil de pénalité maximal à appliquer.

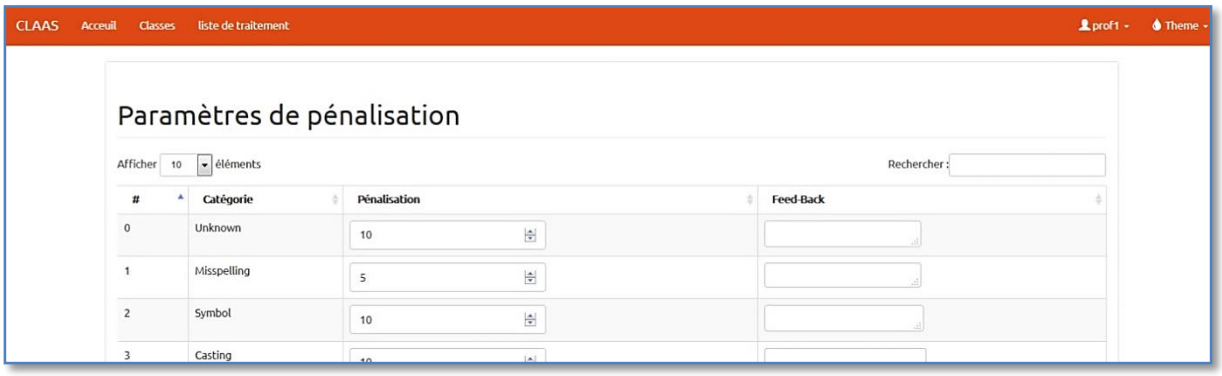

Figure 31 : Compte enseignant - Options de pénalisation

Après avoir appliqué, au programme évalué, toutes les corrections possibles dans CLAAS, le calcul de la note finale se fait suivant l'expression suivante :

Note = Note maximale – pénalité

Tel que :

Pour chaque catégorie d'erreurs corrigées :

Pourcentage pénalité = pourcentage de la catégorie * Nombre d'occurrence

Si pourcentage pénalité > pourcentage pénalité maximal

pénalité = Note maximale * pourcentage de pénalité maximal

Sinon

pénalité = Note maximale * pourcentage pénalité

Les figures suivantes illustres les résultats d'évaluation de l'exemple présenté.

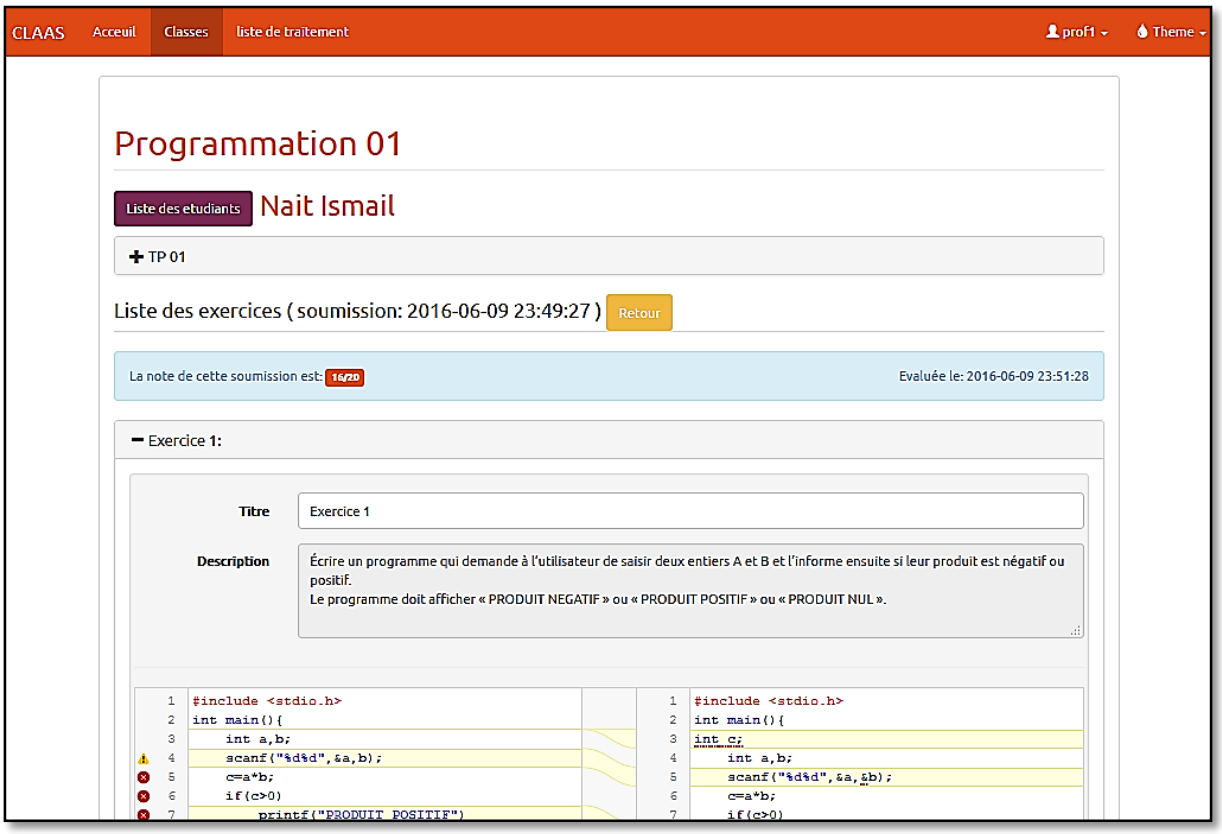

Figure 32 : Compte enseignant - Affichage d'une soumission

Après l'évaluation, l'enseignant peut visualiser les détails de la correction d'une soumission.

Si un exercice a été corrigé, l'affichage comparatif est présenté avec les détails des erreurs…

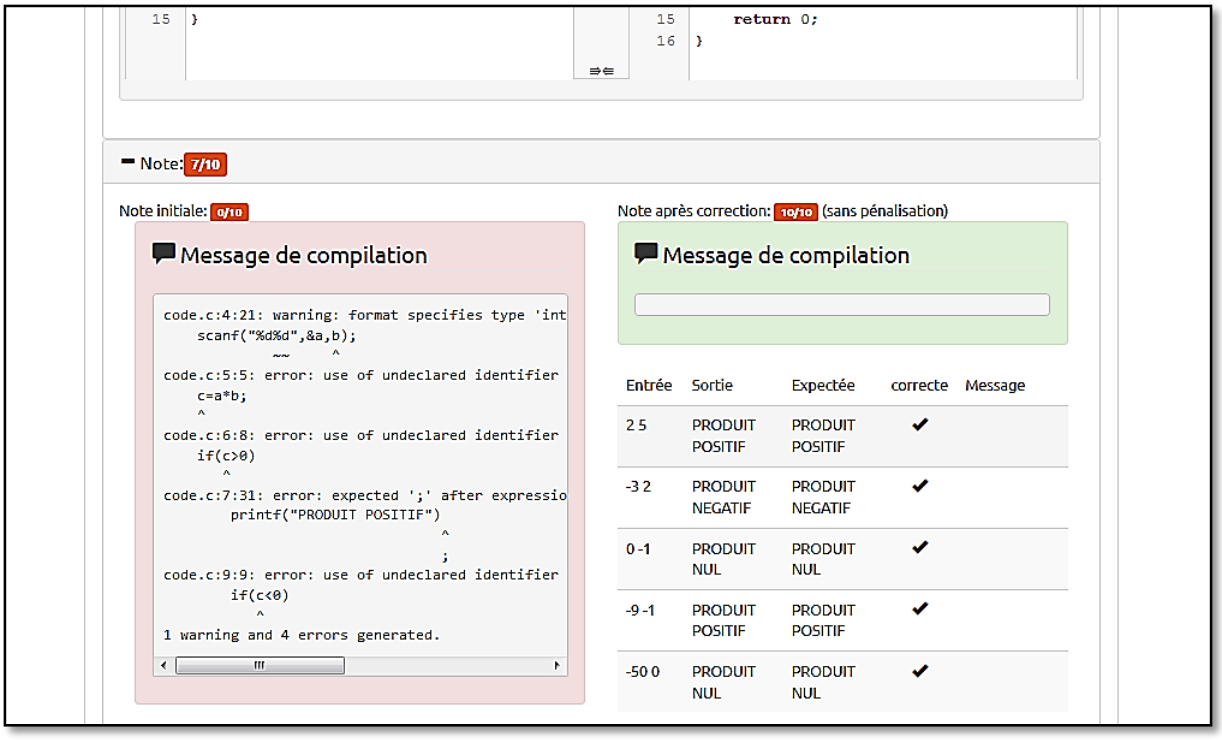

Figure 33 : Compte enseignant - Résultat de l'évaluation (cas de correction)

Les résultats de l'évaluation du programme sont affichés en dessous du code soumis, à savoir, la note attribuée, les messages de compilation et les résultats des cas de test.

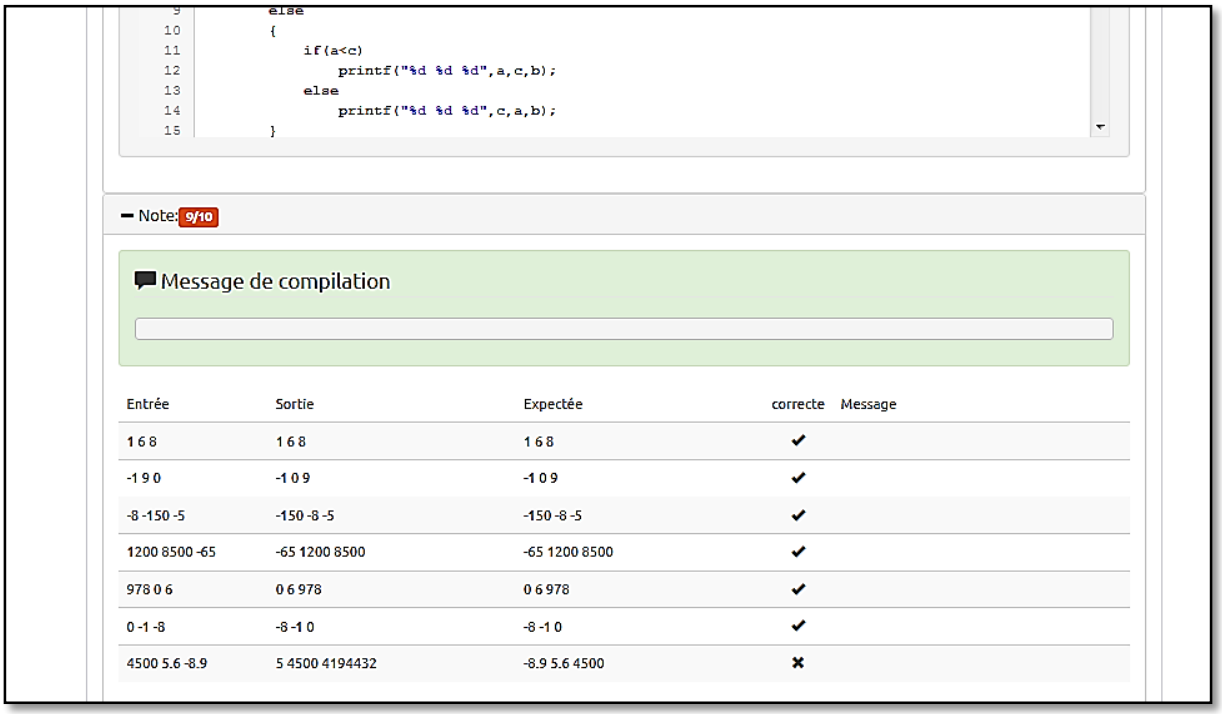

Figure 34 : Compte enseignant - Résultat de l'évaluation (cas sans erreurs)

Chapitre4. Implémentation de l'unité de détection et de correction d'erreurs

I. Choix du compilateur CLANG

« *Clang est un compilateur pour les langages de programmation C, C++ et Objective-C. Son interface de bas niveau utilise les bibliothèques LLVM pour la compilation.*

C'est un logiciel libre issu d'un projet de recherche universitaire et distribué selon les termes de la licence Open Source NCSA/Université de l'Illinois.

Clang est aujourd'hui maintenu par une communauté autour de Chris Lattner chez Apple dans le cadre du projet LLVM. Son but est de proposer une alternative à GCC. » - Wikipedia

Clang est en effet un compilateur rapide et utilise moins de mémoire. Il présente de nombreux autres avantages par rapport à d'autres compilateurs [14]. Il est conçu pour retenir plus d'informations au cours du processus de compilation et préserve la forme globale du code original. L'objectif de cela est de rendre plus facile le mappage de erreurs dans le code d'origine.

En outre, les messages d'erreur offerts par Clang sont destinés à être plus détaillés, précis et expressif, ainsi que lisible par machine. Cela présente un énorme avantage pour faciliter la localisation des erreurs lors de la correction.

Dans notre contexte, Clang a été utilisé comme compilateur principal en raison des avantages énumérés.

Voici quelques options de compilation utilisées :

-Weverything : Activer tous les diagnostics.

-fdiagnostics-parseable-fixits : Afficher les indice de correction sous forme lisible par machine.

-fdiagnostics-show-category=name : Afficher le nom de la catégorie de l'erreur.

-fdiagnostics-print-source-range-info : Afficher les plages d'erreur sous forme lisible par machine.

Exemple :

```
code.c:5:18:{5:15-5:18}{5:19-5:20}: error: invalid operands to binary
expression ('char *' and 'int') [Semantic Issue]
        P = (P-42) + nom*4;
                     ~~~^~
```

II. Expérimentation préliminaire

L'intégration de l'unité de correction dans le système CLAAS, donne et sans aucun doute une valeur ajoutée au processus de l'évaluation automatique.

Malgré les contraintes de temps nous avons réussi à intégrer la correction automatique pour de nombreuses erreurs. On note que le système reste extensible pour la prise en charge d'autres erreurs. L'exemple suivant est présenté afin d'illustrer le processus de détection et de correction automatique intégré dans CLAAS.

Soit le code erroné suivant :

```
1  #include<stlib.h>
2  int main(){
3
4      do{
5          scanf("%d",x);
6      }while(x<0)
7
8      y=y+x;
9      switch y{
10         case 1 y=5;break;
11         default: y=4;
12     }
13
14     if((x%2=0)
15         ptintf("hello world %d",x);
16     els
17         printf("inpaire);
18
19     return 0;
20 }
```

Figure 35 : Exemple de code à corriger

```c
#include<stdio.h>
/*#include<stlib.h>*/
int main(){
int y = 0;
int x;

    do{
        scanf("%d",&x);
    }while(x<0);

    y=y+x;
    switch (y){
        case 1: y=5;break;
        default: y=4;
    }

    if((x%2==0)
        )printf("hello world %d",x);
    else
        printf("inpaire");

    return 0;
}
```

Figure 37 : Le code corrigé automatiquement par CLAAS

```
15 erreurs corrigées! (6 iterations)

Catégories des erreurs:
Orthographe (3)
Symbole (5)
Casting (1)
Bibliothèque (1)
Logic (1)
Variable (3)
Boucle (1)
```

Figure 36 : Les catégories des erreurs trouvées dans le programme

CLAAS permet aussi à l'utilisateur d'avoir une vue comparative du code avant et après correction des erreurs:

```c
#include<stlib.h>
int main(){

    do{
        scanf("%d",x);
    }while(x<0)

    y=y+x;
    switch y{
        case 1 y=5;break;
        default: y=4;
    }

    if((x%2=0)
        ptintf("hello world %d",x);
    els
        printf("inpaire);

    return 0;
}
```

```c
#include<stdio.h>
/*#include<stlib.h>*/
int main(){
int y = 0;
int x;

    do{
        scanf("%d",&x);
    }while(x<0);

    y=y+x;
    switch (y){
        case 1: y=5;break;
        default: y=4;
    }

    if((x%2==0)
        )printf("hello world %d",x);
    else
        printf("inpaire");

    return 0;
}
```

Figure 38 : Vue comparative du code

Afin de tester la performance de CLAAS après l'intégration de la détection et de la correction automatique des erreurs. Nous avons effectué une expérimentation sur des programmes soumis par 4 étudiants durant la colle de TP en programmation C sur quatre étudiants de la première année BCG.

Voici un tableau résumant les résultats :

Etudiant	Note (sans correction)	Note (avec correction)	Note manuelle
1	7.25	9.25	10
2	13	13	17
3	5	8.6	14
4	14.5	14.5	15

Tableau 2 : Notes avant et après l'intégration de la détection et la correction d'erreurs dans CLAAS

On remarque que l'intégration de la détection et de la correction automatique des erreurs dans CLAAS améliore considérablement les résultats de l'évaluation tout en sachant que la base d'erreurs corrigibles actuellement est encore limitée. La performance de CLAAS serait de mieux en mieux avec l'implémentation de plus en plus d'erreurs dans la base d'erreurs.

Conclusion et perspectives

L'évaluation manuelle des épreuves de programmation est une tâche fastidieuse pour les enseignants surtout pour les cours d'initiation à la programmation où l'effectif des étudiants est important. L'intégration des systèmes d'évaluation automatiques des programmes contribuera sans doute à la réduction de la charge de correction et d'évaluation manuelle des programmes qui s'avère couteuse en termes de temps.

L'avantage majeur de la méthode d'analyse dynamique réside dans la facilité d'implémentation. En effet, le fruit de ce travail était la réalisation du système d'évaluation CLAAS basé sur la méthode d'analyse dynamique.

Après les expérimentations effectuées de CLAAS, un problème a remonté à la surface, le processus de l'évaluation étant interrompu dans le cas de présence d'erreurs dans le programme évalué. La méthode d'analyse dynamique qui considère le programme tel une boite noire et ne s'intéresse qu'aux sorties générées et ne peut donc pas évaluer un programme qui ne s'exécute pas, résultant à des notes très défavorables aux étudiants.

Pour remédier à ce problème on a intégré, dans CLAAS, une unité de détection et de correction automatique de programme dans le but de garantir au maximum son évaluation.

Les résultats préliminaires semblent satisfaisants, et le système reste extensible pour assurer la correction de plus d'erreurs.

Cependant, le système reste impuissant devant les erreurs logique vu leurs invisibilité au compilateur d'une part, et la difficulté de les corriger d'autre part. Ce dernier problème pourrait être le sujet d'une autre recherche, traitant l'intégration de l'analyse statique des programmes dans CLAAS pour la détection des erreurs.

Egalement, en guise de perspectives, l'analyse statique par métriques et par mots clés pourrait être intégrée dans CLAAS afin de :

- Accéder au code source du programme afin de renseigner sur le respect des exigences de l'évaluateur.
- Présenter par les métriques, un tableau de bord affichant des statistiques servant à guider l'évaluateur dans ses orientations et ses actions.

Références

[1] Fonte, Daniela, Daniela da Cruz, Alda Lopes Gançarski, and Pedro Rangel Henriques. "A Flexible Dynamic System for Automatic Grading of Programming Exercises." In OASIcs-OpenAccess Series in Informatics, vol. 29. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[2] Mandal, Amit Kumar, Chittaranjan Mandal, and Chris Reade. "A system for automatic evaluation of programs for correctness and performance." In Web Information Systems and Technologies, pp. 367-380. Springer Berlin Heidelberg, 2007.

[3] Singh, Rishabh, Sumit Gulwani, and Armando Solar-Lezama. "Automated feedback generation for introductory programming assignments." In ACM SIGPLAN Notices, vol. 48, no. 6, pp. 15-26. ACM, 2013.

[4] Rahman, Khirulnizam Abd, and Md Jan Nordin. "A review on the static analysis approach in the automated programming assessment systems." In Proceedings of the national conference on programming, vol. 7. 2007.

[5] Hollingsworth, Jack. "Automatic graders for programming classes."Communications of the ACM 3, no. 10 (1960): 528-529.

[6] Forsythe, George E., and Niklaus Wirth. "Automatic grading programs."Communications of the ACM 8, no. 5 (1965): 275-278.

[7] Hext, Jan B., and J. W. Winings. "An automatic grading scheme for simple programming exercises." Communications of the ACM 12, no. 5 (1969): 272-275.

[8] Ihantola, Petri, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. "Review of recent systems for automatic assessment of programming assignments." InProceedings of the 10th Koli Calling International Conference on Computing Education Research, pp. 86-93. ACM, 2010.

[9] Xavier, J., and A. Coelho. "Computer-based assessment system for e-learning applied to programming education." ICERI2011 Proceedings (2011): 3738-3747.

[10] Matt, Urs von. "Kassandra: the automatic grading system." (1998).

[11] Edwards, Stephen H., and Manuel A. Perez-Quinones. "Web-CAT: automatically grading programming assignments." In ACM SIGCSE Bulletin, vol. 40, no. 3, pp. 328-328. ACM, 2008.

[12] Hristova, M., Misra, A., Rutter, M., & Mercuri, R. (2003, February). Identifying and correcting Java programming errors for introductory computer science students. In ACM SIGCSE Bulletin (Vol. 35, No. 1, pp. 153-156). ACM.

[13] Campbell, J. C., Hindle, A., & Amaral, J. N. (2014, May). Syntax errors just aren't natural: improving error reporting with language models.

[14] Guntli, C. (2011). Architecture of clang. Analyze an open source compiler based on LLVM.

Annexes

Annexe 1

Questionnaire pour l'évaluation de l'outil de soumission automatique de programmes en langage C

Durant la présente séance de TP, vous avez eu l'occasion de tester un outil en cours de création dédié à l'évaluation automatique de programme écrit en langage C.

Nous vous prions de bien vouloir répondre aux questions suivantes afin de nous aider à assurer le bon fonctionnement de cet outil.

Question n°1. L'utilisation de l'outil est :

☐ Très facile ☐ Facile ☐ Difficile ☐ Très difficile

Question n°2. Avez-vous eu besoin d'une assistance pour l'utilisation de cet outil ?

☐ Oui ☐ Non

Si oui, veuillez indiquer le nombre d'intervention d'un assistant au cours de ce TP.

Question n°3. Avez-vous déjà eu l'occasion d'utiliser un outil pareil ?

☐ Oui ☐ Non

Si oui, veuillez préciser le contexte de cette utilisation.

Question n°4. La qualité de l'interface de l'outil est :

☐ Très bonne ☐ Bonne ☐ Moyenne ☐ Mauvaise ☐ Très mauvaise

Question n°5. Envisagez-vous l'utilisation de cet outil durant l'examen ?

☐ Oui ☐ Non

Question n°6. Préférez-vous la méthode traditionnelle (la soumission des programmes sur un support papier) à la méthode numérique (l'utilisation d'un outil de soumission automatique)

☐ Oui ☐ Non

Question n°7. Envisagez-vous que vos programmes soient corrigés et notés par un outil d'évaluation automatique ?

☐ Oui ☐ Non ☐ Indifférent

Annexe 2

Outils de développement de l'application

Le système développé est une application client-serveur basée sur les technologies web actuelles :

- **PHP 5** (Hypertext Preprocessor) : Plus connu sous son sigle PHP, est un langage de programmation compilé à la volée libre principalement utilisé pour produire des pages Web dynamiques via un serveur HTTP, mais pouvant également fonctionner comme n'importe quel langage interprété de façon locale. PHP est un langage impératif disposant depuis la version 5 de fonctionnalités de modèle objet complètes.

- **HTML** (Hypertext Markup Language), C'est le format de données conçu pour représenter les pages web. C'est un langage de balisage permettant d'écrire de l'hypertexte, d'où son nom. HTML permet également de structurer sémantiquement et de mettre en forme le contenu des pages, d'inclure des ressources multimédias dont des images, des formulaires de saisie, et des programmes informatiques. Il permet de créer des documents interopérables avec des équipements très variés de manière conforme aux exigences de l'accessibilité du web.

- **CSS** **Cascading** *Style Sheets* (feuilles de styles en cascade) : servent à mettre en forme des documents web, type page HTML ou XML. Par l'intermédiaire de propriétés d'apparence (couleurs, bordures, polices, etc.) et de placement (largeur, hauteur, côte à côte, dessus-dessous, etc.), le rendu d'une page web peut être intégralement modifié sans aucun code supplémentaire dans la page web. Les feuilles de styles ont d'ailleurs pour objectif principal de dissocier le contenu de la page de son apparence visuelle.

- **JavaScript** (souvent abrégé JS) est un langage de programmation de scripts principalement utilisé dans les pages web interactives mais aussi côté serveur. C'est un langage orienté objet à prototype, c'est-à-dire que les bases du langage et ses principales interfaces sont fournies par des objets qui ne sont pas des instances de classes, mais qui sont chacun équipés de constructeurs permettant de créer leurs propriétés, et notamment une propriété de prototypage qui permet d'en créer des objets héritiers personnalisés.

- **jQuery** est une bibliothèque JavaScript libre qui porte sur l'interaction entre JavaScript (comprenant **Ajax**) et HTML, et a pour but de simplifier des commandes communes de JavaScript. La première version date de janvier 2006.

- **Twitter Bootstrap** est une collection d'outils utile à la création de sites web et applications web. C'est un ensemble qui contient des codes HTML et CSS, des formulaires, boutons, outils de navigation et autres éléments interactifs, ainsi que des extensions JavaScript en option. C'est l'un des projets les plus populaires sur la plate-forme de gestion de développement GitHub.

- **MySQL** est un système de gestion de base de données (SGBD). Il est distribué sous une double licence GPL et propriétaire. Il fait partie des logiciels de gestion de base de données les plus utilisés au monde1, autant par le grand public (applications web principalement) que par des professionnels, en concurrence avec Oracle, Informix et Microsoft SQL Server.