Présentation oral

		Pierre	Thibaut
1	Page de garde	Changer de page dès qu'on va commencer à parler	
2	Sommaire	Nous allons dans un premier temps vous présenter le sujet de notre projet, preuve de prog en log de Hoare, puis on verra les objectifs du projet, ensuite on s'intéressera à la logique de Hoare, puis on vous présentera le travail qu'on a effectué pour terminer par une conclusion.	
3	Présentation du sujet	1) Lire le sujet	2) Donc concrètement, c'que ça veut dire, c'est qu'on a programme (relativement basique, dans notre cas), et qu'on veut pouvoir vérifier que ce programme est correct, sans même devoir l'exécuter. Ce sujet est une introduction à la preuve de programme en logique de Hoare.
3	Objectifs du projet	1) On s'est donné comme objectifs de s'initier à la construction de preuve de programme grâce à la méthode de Hoare, d'automatiser la vérification d'une preuve construite avec la méthode de Hoare et de créer un assistant de construction de preuve lié au vérificateur.	2) Au lieu d'automatiser une construction de preuve (trop complexe), on s'est donné comme objectif d'automatiser un vérificateur de preuve construite avec Hoare.
5	Logique de Hoare - Généralités	1) De manière générale, un programme doit respecter les critères suivants : Terminaison, Correction, Complexité, (la correction et la complexité impliquent la Terminaison) La façon de raisonner sur la correction : pour montrer qu'un algorithme est correct, il s'agit de montrer que quels que soient les paramètres donnés, l'algorithme retourne ce qu'on attend de lui. Pour cela, on découpe le programme en blocs d'instruction élémentaire On raisonne en le réécrivant avec le formalisme donné par la méthode de Hoare, formalisme composé de règles et triplet. La méthode de Hoare offre l'avantage de prouver formellement des programmes sans avoir à les exécuter	2) Définition règle : forme de décomposition syntaxique, prémisse de 0 à 2 triplets, conclusion d'un triplet. Faire référence aux règles données dans le rapport papier. Définition triplet : composé d'une précondition, d'un programme, et d'une postcondition. La précondition décrit l'état du système avant l'exécution du programme, et la postcondition décrit l'état postérieur à l'exécution du programme.
6	Logique de Hoare - Construire une preuve		Maintenant on va suivre ensemble la construction d'une preuve en utilisant quelques règle de Hoare. Tout d'abord, il faut convertir le programme que nous voulons prouver en triplet, appelé racine de l'arbre de preuve. Pour cela on doit repérer ou définir la précondition, le programme et la postcondition. Ici, il n'y a pas de précond spécifiée donc on considère que l'état initial du système est valide, le programme est composé de deux instructions d'affectation et la postcond est explicitée.
7		A ce triplet on applique une règle de Hoare, en l'occurrence la règle de la séquence, qui créé	

		deux nouvelles branches à l'arbre de preuve. La difficulté ici est de trouver le prédicat commun aux deux prémisses générées. Pour s'aider on doit anticiper la règle de l'affectation qu'on appliquera à la deuxième prémisse.	
8			En continuant d'appliquer les règles de Hoare, on arriver à terminer l'arbre de preuve et donc à prouver la correction du programme car les triplets sont valides. On a dû utiliser la règle de la conséquence parce qu'on ne peut en appliquer aucune autre, on a réussi à générer une nouvelle précondition qui nous a permise d'appliquer la règle de l'affectation.
9	Logique de Hoare - Vérifier une preuve	L'objectif d'un vérificateur serait donc d'analyser automatiquement les règles au niveau syntaxique, et automatiquement les triplets au niveau sémantique. Donc le but c'est de vérifier que la preuve est correctement construite selon les spécifications de la logique de Hoare pour au final raisonner sur la correction du programme global que l'on a décomposé.	
11	Travail effectué - Format de preuve	1) Maintenant on passe au travail effectué	2) On devait faire un vérificateur de preuve. On doit passer une preuve à notre programme, et il nous répond si la preuve est correctement construite ou pas. On alors réfléchi à deux formats de preuve : le premier choix était un format en arbre, comme la construction d'une preuve à l'écrit. Le deuxième choix était un format en ligne, ressemblant à une suite de fonctions imbriquées : plus simple à implémenter car on a un nom de règle comme nom de fonction, suivit d'un triplet, suivit de 0 à 2 fonctions en arguments Même si le deuxième format de preuve peut sembler peu intuitif, il nous semblait plus agréable à exploiter que le premier : l'analyse syntaxique se ferait de façon structurée et récursive, et la preuve s'analyserait plus facilement que si elle était écrite dans le premier format (car la lecture se ferait uniquement de gauche à droite)
12	Travail effectué - Analyseur syntaxique	1) Une fois le format de preuve choisi, il a fallu déterminer les outils que l'on allait utiliser pour coder notre vérificateur. Concevoir son propre outil d'analyse implique de devoir coder son analyseur sémantique. On a étudié les analyseurs syntaxiques existants. On a finalement opté pour Flex qui est un outil de génération d'analyseurs lexicaux et GNU Bison qui est un outils de génération d'analyseur syntaxique. 3) Et puis enfin, pour mettre en place l'analyse sémantique, on a fait des structures en C pour stocker, en plus des caractères, la valeur entière (ou booléenne) de certains lexèmes et	2) On a étudié en détails le fonctionnement de l'analyseur syntaxique : découpe une chaîne de caractères lue en fonction des lexèmes définis (c'est l'analyse lexicale), puis analyse la suite de lexèmes obtenus en fonction des règles récursives définies dans la grammaire de l'analyseur syntaxique. Donc, en plus de la prise en main du langage Yacc, on avait à définir cette grammaire, il fallait repérer les lexèmes présents dans une preuve et créer les règles de vérification.

		certaines règles de la grammaire de l'analyseur.	
13	Travail effectué - Interface graphique	2) On part d'une racine, avec un champ texte vide et un widget au dessus qui nous permet d'ajouter une règle. Lorsque on ajoute une règle, ça ajoute le nombre de champs texte nécessaires à la construction de l'arbre. Exemple, une règle seq ajoute deux champs texte et deux widgets AjouterRègle, une règle Conseq n'ajoute qu'un champ et qu'un widget AjouterRègle. Et une règle AFF termine l'arbre, comme indiqué sur l'image. Une fois la preuve construite par l'utilisateur, on valide avec le bouton vert en haut à gauche, et s'il y a erreurs, elles s'affichent en bas de l'arbre en indiquant s'il s'agit d'erreurs sémantiques ou syntaxiques, et en essayant de situer l'erreur.	1) Une fois l'analyseur terminé, nous avons commencé à réfléchir à l'interface graphique. Les objectifs étaient d'améliorer l'affichage du résultat que l'on obtenait avec le terminal, et aussi améliorer la construction de preuve pour la rendre plus intuitive (en forme d'arbre donc : comme la construction écrite) 3) Nous avons aussi la possibilité de remette à zéro la preuve, d'ouvrir la dernière preuve qui a été validée, copier une preuve au format linéaire. On a aussi un menu en haut de la fenêtre avec un onglet d'aide, un exemple de construction de preuve, un onglet à propos, et tout ça.
14	Conclusion	2) Enfin, cette expérience nous a appris à travailler ensemble et à nous partager les tâches d'un même projet.	1) Pour conclure, la mise en place d'une interface graphique intuitive permet de proposer un assistant de construction de preuve et de l'analyser, sans passer par le terminal. Ce projet nous a apporté des connaissances supplémentaires dans des domaines étudiés durant notre troisième année de licence. Nous nous sommes initiés à la logique de Hoare et nous avons eu l'occasion de découvrir les outils d'analyse de langage tels que Flex et GNU Bison.

Readme pour la compilation dans le git pour le prof

Être calé sur les choses dont on ne parle pas dans le diapo :

- Variant / Invariant de boucle et Correction totale / Partielle
- La logique de Hoare sert à décomposer un programme, pour raisonner sur sa correction (= le résultat donné par l'algo est celui attendu)

Définitions à connaître :

- **Intro à la logique de Hoare** : L'inférence est un raisonnement permettant de passer d'une ou plusieurs assertions (= énoncé vrai, prémisse vraie, pré-condition) à une nouvelle assertion qui en est la conclusion.
- **Fonctionnement d'un analyseur syntaxique** : Une grammaire est un formalisme permettant de définir une syntaxe (et par extension un langage formel), c'est-à-dire un ensemble de mots admissibles sur un alphabet donné.