Projet 42sh

pierra_j
macher_v
tranca_p
wipf_a
yeh_j

Novembre 2012

Table des matières

Chapitre 1

Introduction

Le projet 42sh, probablement le projet le plus connu de l'ING1 à l'EPITA, consiste en la création d'un shell de type Bourne, à partir du langage C. Ce projet se déroulant sur quatres semaines est à réalisr par groupe de cinq élèves. Le groupe présentant ce rapport de projet se constitue de Johann Pierrat, Valentin Macheret, Philippe Tranca, Jimmy Yeh et Maxime Wipf.

Le projet 42sh est sans doute le plus bénéfique parmi tous ceux réalisés au premier semestre d'ING1, tant au niveau de l'apprentissage du langage C qu'au niveau de l'environnement UNIX. De plus, ce projet informatique a également pour objectif d'enseigner la gestion de notre temps sur un projet à moyen et long termes ainsi que le travail en groupe. Ce projet n'a pas uniquement pour but de conclure la partie C/UNIX de l'EPITA, il permet une ouverture d'enseignement sur de nouveaux langages de programmation haut niveau comme le Python, le Perl ou le Ruby qui nous seront utiles afin d'assurer le bon fonctionnement de ce projet.

Afin d'expliquer au mieux la planification et la démarche à suivre de la réalisation de ce projet, ce rapport a été divisé en deux grandes parties. La première abordera la découpe en parties du 42sh afin de faciliter sa gestion. La seconde partie précisera le rôle de chaque membre dans le projet ainsi que les temps de réalisation prévus.

Chapitre 2

Présentation du groupe

2.1 Johann Pierrat

Etudiant à l'EPITA depuis cette année, j'ai depuis le début de la piscine appris à changer ma façon de travailler afin de m'habituer au rythme de travail de l'école. Ces nouvelles méthodologies m'ont aidé pour les différents projets. Le projet 42sh va me permettre de tester ma gestion personnelle, ainsi que mes connaissances sur le C et UNIX.

2.2 Valentin Macheret

Etudiant à l'Epita depuis l'info-sup, l'arrivée en ING1 a engendré de nombreux changements dans mes méthodes de travail. Cette méthodologie et organisation m'ont permis de suivre les différents travaux pratiques proposés par l'école, seul comme en groupe. Cependant, le 42sh, sera d'une toute autre ampleur, ce projet me permettra de mettre en pratique toutes mes connaissances en C tout en ayant a gérer au mieux mon temps et mon organisation personnelle.

2.3 Philippe Tranca

Etudiant également à l'EPITA depuis l'année de Sup, j'ai pu apprendre de nombreuses choses sur la programmation, qui m'ont très largement servies pendant cette tant attendue piscine. N'étant pas étranger aux projets de groupe grâce à l'expérience acquise à l'EPITA, j'espère réussir 42sh aussi bien que les autres projets que j'ai pu mener. J'espère que ce projet m'apportera de nombreuses connaissances en C tout comme les précédents ont pu le faire avec leurs langages respectifs.

2.4 Maxime Wipf

Etudiant à l'EPITA depuis l'année de Sup, j'ai dans mes connaissances une certaine gestion de groupe lors de différents projets à moyen et long termes. J'espère pouvoir, lors de ce projet, mettre à disposition ces connaissances, ainsi que les approfondir. Par ailleurs, ce projet boostera de manière non négligeable mes connaissances sur le C et UNIX ; ces dernières étant, de loin, non complètes.

2.5 Jimmy Yeh

Le 42sh est pour tout étudiant Epita, le projet qui leur permettra d'approfondir ses connaissances en C. N'ayant fait que de petits projets depuis le début de l'année, le 42sh est le premier projet important qui demandera un vrai travail d'équipe, de l'organisation. C'est avec un esprit positif que je débute ce projet.

Chapitre 3

Découpage et conception du projet

Pour ce projet, nous avons décidé de nous séparer en plusieurs sous-groupes afin de répondre aux besoins du projet. Pour cela nous avons affecté à chacun une partie du travail à effectuer, ce qui permet de réaliser plusieurs objectifs en parallèle sans avoir à attendre qu'une partie du projet soit terminée afin de pouvoir poursuivre le reste du travail.

Nous nous sommes beaucoup inspirés du sujet concernant la séparation des tâches. Elle s'est en effet faite en fonction des différentes versions à réaliser, afin d'avoir une séparation logique dans les objectifs demandés.

3.1 Outils utilisés

Afin d'écrire notre code, nous utiliserons des éditeurs de texte classiques tels que Emacs ou Vim. Pour compiler notre projet, l'outil utilisé sera la suite de compilateurs GCC. De plus, l'outil Cmake sera utilisé pour la génération des makefiles. Enfin, le gestionnaire de versionnement de ce projet sera Git.

3.2 Version 0.1

Pour la version 0.1, il nous est demandé trois documents bien distincts :

- Une page de man sur notre shell afin de décrire son utilisation à l'utilisateur lambda.
- Un cahier des charges (vous le lisez en ce moment même d'ailleurs) de 20 pages maximum afin de bien définir les objectifs à atteindre et comment nous allons réussir à les atteindre.
- Un powerpoint de présentation.

3.3 Version 0.5

Pour la version 0.5, nous devons implémenter les fonctions de base d'un shell, mais aussi des fonctionnalités qui facilitent l'avancée de ce dernier. Par ordre d'importance, les

objectifs à réaliser sont : le Makefile (avec CMake), le lexer, le parser, l'AST, le prompt, l'exécution, et enfin la moulinette de tests.

3.3.1 Le Makefile

Le Makefile sera réalisé avec CMake et aura pour but de répondre aux exigences du sujet, c'est à dire générer l'exécutable du projet mais aussi implémenter les autres règles classiques (clean, distclean, doc, check).

3.3.2 Le lexer

Le lexer (ou analyseur lexical) consiste à décomposer une chaîne de caractères en unités ou entités lexicales, on peut également appeler celles-ci "tokens". Ces entités lexicales, sont générées sous la demande du parser qui les consomme par la suite. Le parser est défini par un ensemble de règles, généralement appelées expressions rationnelles.

3.3.3 Le parser

Le parser analyse ce qu'il a demandé au lexer et vérifie que l'expression répond correctement à la norme ainsi qu'à la grammaire. Il le transforme par la suite en AST.

Pour cette version, le développement du parser se décompose en deux étapes :

- La gestion des options du parser.
- la construction de l'AST. Nous devons créer un AST après avoir parsé du langage shell. Nous utiliserons ici une analyse LL, analyse dite descendante pour un sous-ensemble de grammaire non contextuelle. Celle-ci va analyser l'entrée de gauche à droite et en construit une dérivation à gauche. Afin de réaliser ce parser, nous utiliserons le modèle fourni dans le sujet.

3.3.4 L'afficheur d'AST

L'AST ou abstract syntax tree, est un arbre dont les noeuds sont constitués d'opérateurs et dont les feuilles sont les opérandes de ces opérateurs.

On se contentera pour ce palier d'afficher un fichier “dot”, néanmoins l'affichage de l'AST pour la présentation orale sera également effectué puisque celui-ci est requis pour la présentation orale.

3.3.5 L'exécution

L'exécution consiste à pouvoir exécuter toute la grammaire du shell. C'est à dire que notre shell exécutera les expressions que l'AST lui fournit. L'exécution devra entre autre gérer des opérateurs, des mots clés tels que if ou while, ainsi que des lignes de commandes simples.

3.3.6 La moulinette

La moulinette de tests consiste tout simplement en une série de tests que nous effectuerons afin de vérifier les fonctionnalités de notre shell. Elle sera écrite en Python (langage qu'aucun de nous ne maitrîse) et effecturera des séries de tests pertinents, que l'équipe ajoutera au fur et à mesure de l'avancement du projet.
Le rôle de la moulinette est de tester, et donc éviter les mauvaises surprises mais surtout d'empêcher toute régression dans le code.

Au niveau du fonctionnement de la moulinette, plusieurs catégories de tests sont prévues pour notre projet, chaque catégorie doit pouvoir être lancée séparément. Par conséquent, cette dernière propose également à l'utilisateur différentes options afin de l'affiner.

3.3.7 Le prompt

Le prompt à pour but d'afficher la ligne sur l'écran avant l'entrée d'une commande. Pour cette version nous aurons besoin de PS1 et de PS2.

3.4 Version 0.8

La version 0.8 se caractérise par l'ajout de fonctionnalités plus poussées à notre shell. Par ordre de priorités, nous implémenterons : les builtins, le tilde, un prompt avancé, les variables, gestion des guillemets. le globbing, gestion des expressions arithmétiques, rendre le shell interactif et gérer des alias.

3.4.1 Les builtins

Les builtins sont des commandes présentes dans le shell afin de permettre à l'utilisateur d'effectuer des opérations basiques dans le shell. Pour rappel, les builtins à implémenter sont :

- exit
- cd
- shopt
- export
- alias
- unalias
- echo
- continue
- break
- source
- history

3.4.2 Le tilde

Pour cette version, nous devrons implémenter l'expansion tilde `~`. Utilisé seul, il correspond au "home directory" c'est à dire le chemin absolu vers le répertoire home. Mais on gérera aussi les cas où il n'est pas utilisé.

3.4.3 Un prompt avancé

Dorénavant, le prompt doit pouvoir gérer les variables. De même le quoting doit être réalisable, c'est à dire gérer les ", ', et les ‘, chacun ayant un rôle différent lors de l'interprétation grammaticale du shell.

3.4.4 Les variables

Le shell devra également pouvoir substituer des valeurs aux variables.

3.4.5 Gestion des guillemets

Cette version devra gérer les différents types de quotes : " ", ' ', et ‘ ‘.
- Les simples quotes préservent la valeur littérale.
- Les doubles quotes préservent également la valeur littérale à l'exception des mots précédés par $, ‘ ou \.
- Les backquotes substituent l'expression littérale à sa valeur.

3.4.6 Le globbing

Le shell devra implémenter la gestion des globbing. Le globbing fait référence à la reconnaissance de schémas particuliers. Exemple : "ls *.log" affichera "a.log" et "b.log" dans le cas où nous avons 3 fichiers (a.log, b.log, c.dat).

3.4.7 Les expressions arithmétiques

Les expressions arithmétiques doivent être gérées par le shell pour cette version. C'est à dire que le shell doit ajouter les opérateurs mathématiques tels que +, -, /, *, &, |, ^, &&, ||, !, ~.

3.4.8 Un shell intéractif

Notre shell devra pouvoir être intéractif avec l'utilisateur. Il devra pouvoir gérer les signaux que l'utilisateur envoie, par le biais de combinaisons de touches.

3.4.9 Les alias

Les alias, sont des raccourcis vers d'autres commandes. Ils ont pour but de permettre à l'utilisateur de gagner du temps pour certaines commandes afin de lui éviter de taper la commande en entier (i.e : "e" pour "emacs -nw")

3.5 Version 0.9

Cette version n'étant basée que sur des bonus, nous nous concentrerons essentiellement sur la vérification et l'optimisation de notre code, puis si les conditions sont remplies nous nous intéresserons aux bonus disponibles afin de rendre notre projet plus puissant, attractif et user friendly.

3.6 Version 1.0

Cette version est la version finale du projet. Nous nous consacrerons donc seulement au debugging du projet. Le but étant de n'avoir aucune fuite de mémoire et aucune erreur de segmentation.

Chapitre 4

Répartition et planification des tâches à accomplir

Le projet se déroule environ sur quatre semaines. Afin de gérer au mieux le temps qui nous est imparti, il va de soit qu'une organisation et planification des tâches rigoureuses a été appliquée. Les cinq membres du groupe, Johann Pierrat, Valentin Macheret, Jimmy Yeh, Philippe Tranca et Maxime Wipf, se sont donc séparés les tâches à faire.

Cette partie du rapport met en avant cette répartition tout en abordant les durées prévues pour le développement de chaque élément.

4.1 Version 0.1

Dans ce prélude au 42sh, les tâches à accomplir étaient de réaliser ce rapport faisant office de présentation du projet, la rédaction de la page de manuel, ainsi que la réalisation de slides pour la première présentation orale.
La rédaction du rapport a été divisée en deux, reprenant les deux grandes parties : l'analyse et la découpe du projet par Philippe Tranca et Jimmy Yeh, puis la répartition et planification des tâche par Johann Pierrat et Valentin Macheret. La version latex sera assurée par Maxime Wipf.

La page de manuel pour l'instant remplie du nom, du synopsys, de la description, du copyright et des auteurs du 42sh sera actualisée en fonction du travail accompli. L'objectif étant de posséder une page de manuel (invoquée par "man 42sh" dans le shell) complète sur notre shell. La page du manuel sera actualisée et entretenue par Valentin Macheret.

La première présentation se doit d'être accompagnée par une série de slides permettant de soutenir le groupe pendant son oral. Johann Pierrat et Philippe Tranca se sont concentrés sur cette tâche.

4.2 Version 0.5

Pour cette version, nous devrons inclure un Makefile performant. Pour se faire nous allons utiliser Cmake. Suite aux présentations de lundi dernier, le groupe a choisi de s'étendre sur Cmake afin de générer des makefiles performants et efficaces. Philippe Tranca

se chargera entre autre de la gestion des makefiles via cet outil.

Au lancement du binaire, un parseur d'options devra être intégré. Ce dernier permet de prendre en compte des options dès l'éxécution. Ce travail sera effectué par Johann sur environs deux journées.

Nous devrons également développer un prompt. Ce prompt est une interface de ligne de commande qui invite l'utilisateur à utiliser le shell. Johann Pierrat s'occupera de cette tâche sur une prévision de deux jours.

A partir de ce prompt un parseur de shell doit être fonctionnel et capable de créer un AST et de le remplir en fonction de la ligne de commande rentrée. Cette partie étant très importante et complexe, seront disposés pour sa conception sur quatre à cinq jours : Valentin Macheret, Johann Pierrat et Jimmy Yeh.

Afin de clarifier ce que lit le parseur, il sera implémenté un afficheur d'AST. Cet AST se charge d'établir un ordre sur les éléments de la ligne de commande. Jimmy Yeh se chargera de cette partie, avec une prévision de deux jours.

Enfin, l'exécution, partie très importante, s'occupe d'exécuter toute la grammaire du 42sh. Sur cet élément important, seront disposés Valentin Macheret, Philippe Tranca et Maxime Wipf sur environ trois jours.

Pour tester la qualité et la rapidité d'exécution du code, il va de soit qu'une série de tests sera effectuée. Ces tests seront effectués par une moulinette faite par Maxime Wipf. Les tests seront ajoutés tout au long du développement du projet.

4.3 Version 0.8

Pour le 29 novembre nous devrons rendre une nouvelle version du projet. Dans cette version il faudra avoir un shell basique qui est capable d'effectuer de simple tâches. Pour finir ce shell dans le temps imparti, nous nous sommes séparés le travail à faire en fonction des différents objectifs à accomplir.

Il faudra être capable d'imiter les fonctionnalités présentes sur un bash pour la touche tilde. Pour cela nous avons prévu un travail de deux jours maximum et ce sera Johann Pierrat qui s'occupera du développement. A la fin de ce développement on doit avoir le fonctionnement correct et complet de la touche ~.

L'une des grandes parties de cette version du shell est le développement des différentes builtins existantes. Nous avons estimé qu'une telle tâche nécessite au minimum cinq jours de développement avec au minimum trois personnes travaillant sur cet objectif. Cet objectif va nous permettre d'avoir un shell avec les bases nécessaires pour continuer le développement. Ce sera Jimmy Yeh, Valentin Macheret et Maxime Wipf qui s'occuperont de cette partie.

Dans la version 0.5 nous avons développé un prompt basique qui n'affichait que peu d'informations. Pour cette nouvelle version nous devons ajouter des options supplémen-

taires que l'utilisateur pourra modifier s'il le souhaite. Ce sera Johann Pierrat qui va s'occuper de cet objectif car il avait déjà développé le prompt de base, et donc il sera familier avec ce dernier. Nous avons estimé une charge de travail de deux à quatre jours pour cette partie.

Afin d'avoir un shell convenable nous devons pour le 29 novembre prendre en compte la gestion des variables. Cette partie s'avère assez difficile et nous avons estimé qu'une charge de travail de quatre jours pour deux personnes serait nécessaire. Ce seront Philippe Tranca et Jimmy Yeh qui s'occuperont du développement de cette partie. Cette partie sera nécessaire pour accomplir correctement les calculs arithmétiques.

Valentin Macheret s'occupera de la gestion des simples et doubles quotes. Pour cette partie nous devrons refaire le même fonctionnement que celui présent dans le bash. Nous avons prévus une durée de deux jours pour finir cette tâche.

Afin que les variables développées précédemment soient bien utilisées dans notre 42sh nous devons développer toute l'arithmétique nécessaire pour effectuer de tels calculs. Pour cela nous avons estimé que la charge de travail ne devrait pas dépasser les deux à trois jours car nous avons déjà des connaissances sur cette partie grâce à la piscine. Ce sera Johann Pierrat et Philippe Tranca qui s'occuperont de cette partie.

Pour avoir un meilleur affichage nous devrons développer la partie Readline. Cette partie va nous permettre de prendre en compte les différentes touches que l'utilisateur tape et avoir un affichage plus fluide. On devrait être capable de finir cet objectif en trois jours maximum, avec deux personnes travaillant dessus. Nous avons prévu que Philippe Tranca et Maxime Wipf s'occuperont de cette partie pour tenir les délais de trois jours.

Le dernier module à développer pour cette version est la gestion des alias. Dans cette partie nous devrons prendre en compte les alias définis par l'utilisateur et les utiliser convenablement. Pour accomplir cela nous avons prévus une durée de deux à trois jours avec Maxime Wipf, Jimmy Yeh et Valentin Macheret qui travailleront dessus.

4.4 Version 0.9

Une fois la version 0.8 rendue nous aurons finalisé la partie principale du développement. Nous devrons ensuite pour les versions 0.9 (à remettre le 01/12/2012) et 1.0 (à remettre le 02/12/2012), perfectionner le code existant afin de vérifier qu'il n'y ait aucune erreur présente dans notre programme. Pour cela nous avons séparé le travail à faire en trois parties : le debugging, la vérification mémoire et l'amélioration de notre suite de test si nécessaire.

Pour le debugging nous avons prévu trois personnes travaillant dessus afin de vérifier qu'il n'y ait aucun problème qui pourrait subvenir pendant l'utilisation de notre shell. Cela peut être de simples fautes de segmentation, de mauvaises affectations d'une variable, ou encore d'un alias incorrect. Valentin Macheret, Jimmy Yeh et Philippe Tranca s'occuperont de cette partie.

Il faudra aussi vérifier qu'aucune fuite de mémoire ne soit présente dans notre shell.

Pour cela, Johann Pierrat fera une vérification de la mémoire et corrigera toutes les erreurs présentes.

Maxime Wipf s'occupera de compléter si nécessaire notre suite de tests. Il travaillera avec le debugging pour toute erreur rencontrée.

4.5 Version 1.0

La version 1.0 de 42sh marque la fin de ce projet, la totalité de l'équipe sera chargée d'optimiser le code, de respecter la norme EPITA et de rechercher les problèmes de segmentations et de leaks. La page du manuel sera également mise à jour en fonction de l'état actuel du projet.

4.6 Diagramme de Gantt

Voici un diagramme qui résume l'organisation du projet.

	Task	Assigned To	Start	End	Dur	%
	Start of project		5/11/12	2/12/12	20	14
1	Report	All	5/11/12	7/11/12	3	100
2	Slide	tranca_p	7/11/12	8/11/12	2	100
3	Man	macher_v	5/11/12	6/11/12	2	100
4	Version 0.1		8/11/12	8/11/12		
5	Options parser	pierra_j	10/11/12	12/11/12	1	0
6	Prompt	pierra_j	12/11/12	14/11/12	3	0
7	Shell parser	yeh_j, pierra_j, macher_t	10/11/12	15/11/12	4	0
8	Cmake	tranca_p	12/11/12	14/11/12	3	0
9	AST Printer	yeh_j	16/11/12	17/11/12	1	0
10	Test Suite	wipf_a	10/11/12	13/11/12	2	0
11	Execution	macher_v, tranca_p, wipf_a	16/11/12	18/11/12	1	
12	Version 0.5		18/11/12	18/11/12		
13	Tilde expansion	pierra_j	19/11/12	19/11/12	1	
14	Builtins	yeh_j, macher_v, wipf_a	19/11/12	24/11/12	5	
15	ompt	pierra_j	19/11/12	23/11/12	5	
16	Variable	tranca_p, yeh_j	23/11/12	27/11/12	3	
17	Quoting	macher_v	23/11/12	27/11/12	3	
18	Arithmetic expansions	pierra_j, tranca_p	27/11/12	29/11/12	3	
19	Readline	tranca_p, wipf_a	25/11/12	29/11/12	4	
20	Version 0.8		29/11/12	29/11/12		
21	Debugging	macher_v, tranca_p, yeh_j	30/11/12	2/12/12	1	
22	Memcheck	pierra_j	30/11/12	2/12/12	1	
23	Test Suite	wipf_a	30/11/12	2/12/12	1	
24	Version 0.9		1/12/12	1/12/12		
25	Version 1.0		2/12/12	2/12/12		

Chapitre 5

Conclusion

Le projet du 42sh se promet d'être à la fois attrayant sur un point de vue didactique, car l'ensemble du groupe devra chercher des solutions pour arriver à ses fins, mais également pédagogique, car nous aurons beaucoup à apprendre de ce projet en lui même : la gestion d'équipe, du temps er de la régularité.

Le 42sh permet également une véritable révision des fondamentaux de la programmation en C sous environnement UNIX sans pour autant fermer ce semestre car il est suivi de près par l'apprentissage du C++. Il offrira également l'opportunité à notre groupe d'apprendre de nouveaux langages de programmation haut niveau orientés objets comme le Ruby, le Python ou le Perl.

Enfin ce projet informatique permettra à tous une véritable approche professionnelle en ce qui concerne la gestion de projet en générale : son cahier des charges, son organisation, sa conception, sa mise en avant et enfin, sa vente.