

Méthodologie de la programmation

Système de Gestion de Fichiers
par Lucas Salvato

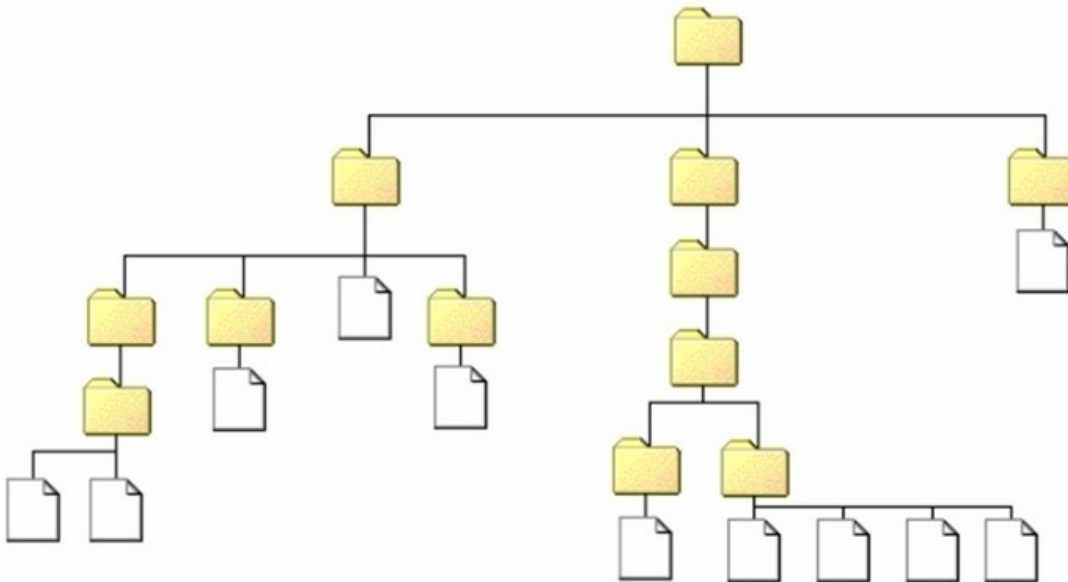


Table des matières

Introduction.....	3
Le Sujet.....	3
Déroulement du projet.....	4
Architecture de L'application.....	5
Principaux Algorithmes.....	6
L'interpréteur de commande.....	6
Renvoyer un nœud à partir d'un chemin.....	6
Le Changement de répertoire.....	6
La Gestion de l'espace mémoire.....	7
La copie d'un répertoire.....	7
Les Tests.....	7
Difficultés rencontrées.....	8
Bilan.....	8
Manuel D'utilisateur.....	9

Introduction

Ce projet est réalisé dans le cadre du cours de Méthodologie de la programmation durant la seconde période 2018, à L'Enseeiht en Informatique et Réseaux par Apprentissage. Il est encadré par M.Yamine Ait-Ameur, M.Neeraj Singh et Mme.Sarah Benyagoud. Le projet a pour objectif d'appliquer l'ensemble des concepts vus en cours de Méthodologie de la programmation.

Vous pouvez retrouver l'intégralité du projet sur Github : <https://github.com/lonaxous/SgfAda>.

Le Sujet

Le but est de simuler un système de gestion de fichiers (Gestion des fichiers et des répertoires d'un ordinateur), avec un mini-interpréteur qui permettra à l'utilisateur d'effectuer des actions en ligne de commande. La mémoire devra aussi être gérée avec une limite fixée à ne pas dépasser.

Tout cela sera réalisé avec le langage de programmation Ada avec l'ensemble des concepts vus en cours de Méthodologie de la programmation pendant la première période 2018.

Dans un premier temps nous allons voir comment le projet s'est déroulé, puis, en deuxième lieu, nous aborderons la composition du programme. Enfin, nous verrons le manuel d'utilisateur.

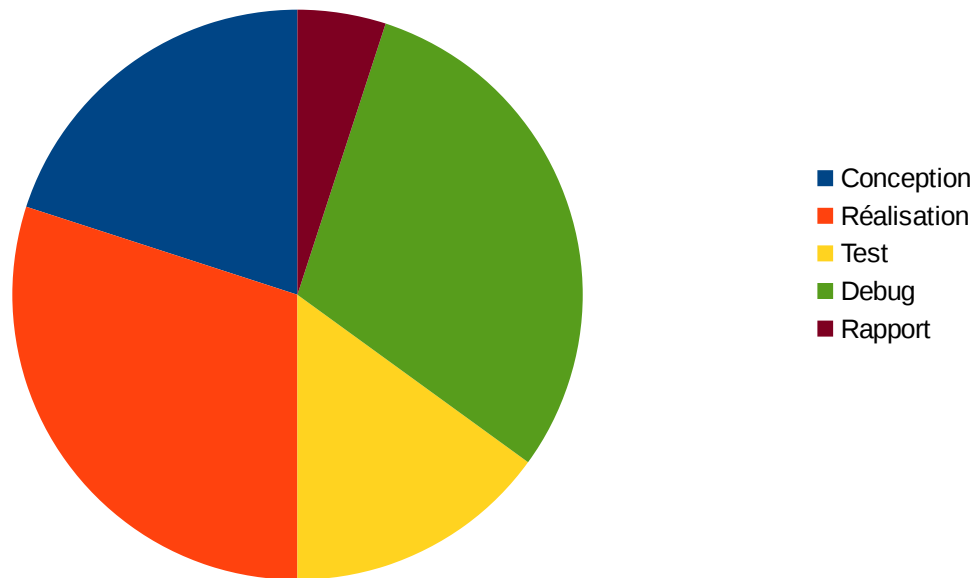


Ada Lovelace

Déroulement du projet

Le projet est réalisé sur une période d'un mois de janvier à février 2018 :

Diagramme de répartition du temps



La conception a pris une part importante du projet étant donné que nous devions rendre toutes les spécifications en premier, celles-ci ont beaucoup changé et, à ce jour, des modules s'y sont rajoutés.

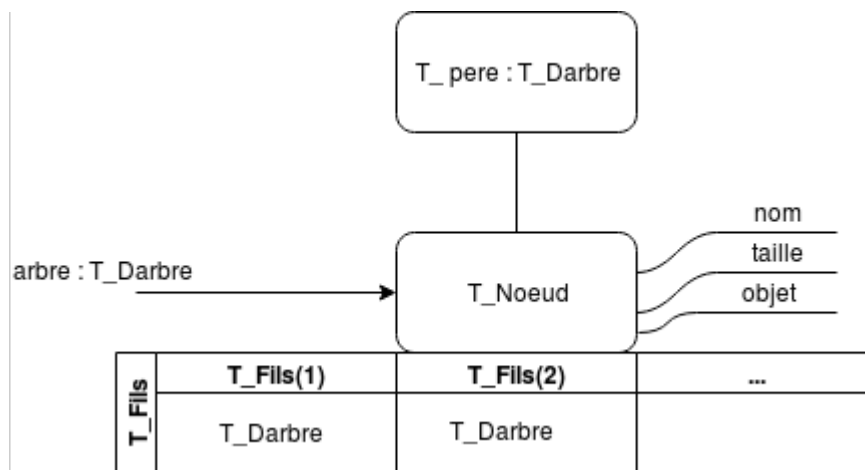
Les tests ont pris une grande place dans le projet, car il fallait penser à tous les cas possibles qui pouvaient mettre en péril le système de gestion de fichiers.

Suite aux tests qui ont révélé un grand nombre d'erreurs, il a fallu les corriger. C'est pour cela que le « débogage » a pris de la place dans le déroulement du projet.

Architecture de L'application

L'application est composée de 3 modules :

- Un arbre générique, permettant de créer une arborescence contenant le type de données de notre choix. Il contient aussi l'ensemble des actions permettant de gérer cet arbre comme par exemple se déplacer dans l'arbre, créer des fils, changer les différentes informations... Un arbre est donc une suite de pointeurs vers des nœuds. Un nœud est composé d'un nom ; d'un pointeur vers un nœud père ; d'un tableau de pointeur vers tous les nœuds fils ; d'une taille ; et d'un objet générique.



Représentation de l'arbre du programme

- Un module SGF va quant à lui implémenter un arbre générique avec un type booléen. Ainsi les nœuds ayant « true » seront considérés comme des répertoires tandis que les autres seront de simples fichiers. Ce module possède aussi son propre arbre afin d'agir directement sur celui-ci suite aux actions effectuées par l'utilisateur.
- Un module miniconsole, qui s'occupe de traiter les demandes entrées par l'utilisateur, les interpréter pour enfin les exécuter dans le SGF.

Principaux Algorithmes

L'interpréteur de commande

L'interpréteur de commande est, de loin, l'algorithme qui a pris le plus de temps à être réalisé de part la complexité des commandes que l'utilisateur peut entrer. Il peut s'agir par exemple de l'utilisation de d'argument de précision comme le « -r ». Une commande est composée de plusieurs arguments, souvent : d'une action (ls, rm, cp, cd, etc.), d'un élément de départ et d'un élément d'arrivée. Ces arguments sont séparés par des espaces. Ainsi, la commande saisie va être analysée lettre par lettre et à chaque espace -sauf pour le premier à cause des précisions d'argument « -r »- nous allons changer d'argument. Ensuite, le programme va vérifier que l'action demandée existe bien et l'exécuter.

(Procédure InterpreteurCommande – miniconsole.adb:35)

Renvoyer un nœud à partir d'un chemin

La commande la plus sollicitée est celle qui, à partir d'un chemin, va déterminer où l'action se situe dans l'arbre. Un chemin est constitué d'une suite de noms de répertoires séparés par des « / » finissant soit par le nom d'un fichier soit par le nom d'un répertoire. Cependant, il y a des exceptions qu'il faut prendre en compte :

- « ../ » Indique le père du répertoire courant.
- « ./ » Indique le répertoire courant.
- Si un chemin commence par « / », alors cela indique la racine de l'arbre.

Ainsi, en premier lieu, l'algorithme va vérifier si le chemin ne commence pas par un « / ». Si cela est avéré, alors le chemin va partir de la racine. Dans le cas contraire, elle va traiter le chemin normalement en vérifiant systématiquement que le nom ou l'exception pointe bien vers une entité existante, et ce jusqu'à arriver à l'endroit indiqué.

(Procédure DetermineChemin – sgf.adb:146)

Le Changement de répertoire

La seconde fonction la plus utilisée permet de changer de répertoire courant à partir d'un chemin. C'est la fonction [DetermineChemin](#) qui va faire tout le travail en renvoyant directement l'arbre indiqué.

(Procédure Cd – sgf.adb:263)

La Gestion de l'espace mémoire

A chaque fois qu'un fichier est créé, supprimé ou modifié. L'espace mémoire doit s'adapter. C'est pour cela qu'il a fallu mettre en place une fonction récursive qui va impacter la taille de tous les pères d'un répertoire jusqu'à la racine.

Cet algorithme va donc parcourir l'ensemble des pères et modifier leur taille en fonction de l'action faite par l'utilisateur. Si l'espace mémoire n'a pas la place d'effectuer cette action, alors cette dernière est annulée. L'espace mémoire est ici limité par une constante.

(Procédure AugmenterTaille – arbre.adb:227)

La copie d'un répertoire

Copier un fichier ou un répertoire contenant des fils n'a pas été une mince affaire. Il faut bien comprendre qu'il ne faut pas que juste créer un pointeur au niveau de la destination mais bien recréer un arbre identique mais avec des pointeurs différents.

La commande [DetermineChemin](#) nous permet de trouver sans trop de difficultés l'objet que nous allons devoir copier et son futur emplacement. Une fonction récursive va alors rentrer en jeu, parcourant l'objet à copier et, pour chaque fils croisé, le recréer dans le répertoire de destination. Le programme va veiller à respecter l'arborescence en revenant à chaque fois vers le pointeur père.

(Procédure CpR – sgf.adb:287)

Les Tests

Les tests sont ici effectués par les programmes tests ([testarbre.adb](#) et [testsgf.adb](#)). Ils vont chacun essayer de mettre en difficulté le système en effectuant des cas d'exceptions comme :

- Supprimer la racine
- Se déplacer vers des répertoires inexistants
- Copier des répertoires déjà bien remplis
- Créer des fichiers avec le même nom
- Etc...

L'objectif est de dénicher les quelques bugs restants et les corriger de la meilleure manière possible.

Difficultés rencontrées

Lors de la réalisation de ce projet en Ada plusieurs difficultés ont été rencontrées :

- La gestion des chaînes de caractères en Ada n'est pas une mince affaire, c'est pour cela que je me suis permis d'utiliser -malgré le fait qu'on ne l'ait pas vu en cours- l' « `unbounded_string` ». Cette librairie permet de créer des chaînes de caractères de taille variable, ce qui simplifie amplement la tâche.
- Penser à l'ensemble des modules en ayant pas encore commencé à programmer était plutôt compliqué. C'est pour cela qu'initialement un seul module était prévu pour réaliser l'entièreté du projet, pour, au final en avoir trois.
- Utiliser la généricité en Ada ne m'a pas semblé simple, c'est après plusieurs essais que j'ai compris comment elle pourrait être utilisée pour ce projet.
- Le projet était dense, j'ai dû modifier, voir supprimer certaines de mes idées de réalisation.
- Le compilateur Ada étant capricieux, j'ai dû renoncer aux variables « `private` », les pré et post conditions -qui sont gérés aujourd'hui par des exceptions à l'intérieur du programme. De plus, j'ai supprimé des « `warnings` » car certains n'avaient pas lieux d'être.


Bilan

Ce projet était éprouvant mais très intéressant. Le fait d'utiliser l'ensemble des compétences vues en cours (les exceptions, la généricité, les pointeurs, la récursivité, etc.) m'a beaucoup appris du langage Ada et ses particularités. En outre, la communauté Ada étant peu présente sur Internet il m'a fallu redoubler d'ingéniosité pour palier aux problèmes rencontrés.

Ada est un langage de programmation avec beaucoup de particularités propres à lui-même. Je vais très probablement m'y retrouver confronté en entreprise dans mon domaine de d'activité. L'aborder en cours va donc m'aider à l'appréhender dans le milieu professionnel.

Manuel D'utilisateur

1. Télécharger le projet à l'adresse : <https://github.com/lonaxous/SgfAda/archive/master.zip>
2. Décompresser l'archive télécharger
3. Se déplacer dans le répertoire extrait
4. Compiler le programme avec la commande : *gnatmake programme.adb*
5. Lancer le programme : *./programme*
6. Nous arrivons alors dans un interpréteur de commande



```
/etc/Documents$ |
```

Le Repertoire Courant

7. Ensuite il vous suffit d'entrée la commande « ? » ou « help » pour accéder à l'ensemble des commandes qui sont interprété par le programme.

```
ls [Chemin Repertoire] | Affiche le contenu du repertoire designé.  
ls -r | Affiche le contenu du repertoire courant et de tous les sous-repertoires  
pwd | Affiche le chemin du repertoire courant.  
touch [Nom fichier] | Créer un fichier dans le repertoire courant.  
mkdir [Nom repertoire] | Créer un repertoire dans le repertoire courant.  
cd [Chemin] | Changement de repertoire.  
rm [Chemin fichier] | Suppression du fichier designé.  
rm -r [Chemin repertoire] | Suppression du repertoire designé.  
mv [Chemin fichier] [Chemin repertoire] | Déplace le fichier designé vers un repertoire donné.  
cp -r [Chemin] [Chemin repertoire] | Copie un élément designé vers un repertoire donné.  
help/? | Affiche la liste des commandes possible.  
capacity | Affiche la capacité restante du disque dur.  
quit | Quitte le SGF.
```