Rapport Projet Méthodologie de la Programmation

Janvier 2026

Rapport préparé par Ellisa EE et Tristan GALLARDO
Enseignant : Neeraj SINGH, Yamine AIT-AMEUR

Contenu

Objectif

L'objectif de ce projet est de développer un système de gestion de fichiers (SGF) permettant de manipuler des fichiers, des répertoires et de gérer l'espace mémoire occupée par ces fichiers. Nous réalisons également deux types d'interfaces d'accès, l'une utilisant un menu et l'autre utilisant des lignes de commande dans un terminal.

Introduction

La gestion de fichier implique non seulement l'organisation et la manipulation des fichiers, mais également une gestion rigoureuse de l'espace mémoire afin d'éviter la perte de données. La problématique abordée dans ce projet est donc la suivante, "Comment développer un SGF permettant de le stockage et la manipulation de données d'une manière efficace et sécurisée ?"

Architecture du projet

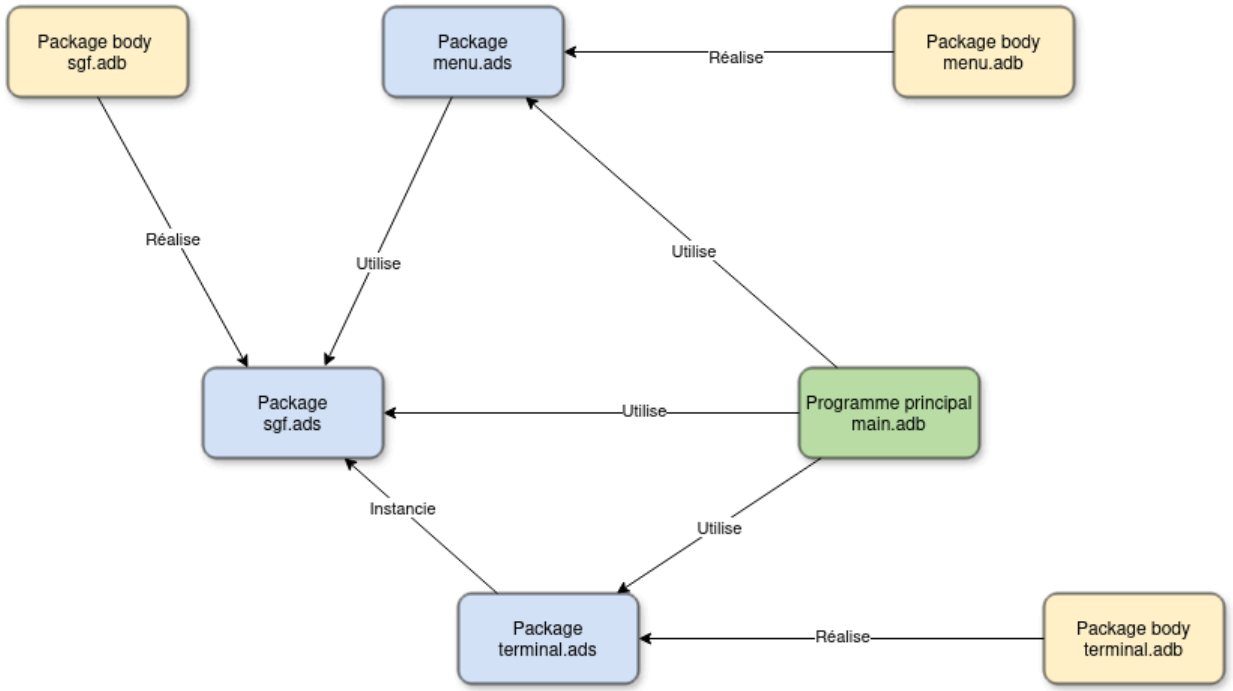

Figure 1 : Architecture logicielle

Principaux choix réalisés

Nous avons choisi d'adopter une approche de programmation défensive, car les données en entrée dépendent de l'utilisateur et sont donc plus susceptibles d'être erronées. Pour cette raison, nous avons utilisé des exceptions afin de gérer les erreurs, en particulier au niveau des interfaces, de manière à éviter que l'exécution du programme ne s'arrête pas en cas d'erreur.

Nous avons également décidé de ne pas inclure la gestion des droits des utilisateurs, bien qu'il soit précisé dans le sujet que chaque fichier est caractérisé par des droits d'accès. Ce choix s'explique par le manque de temps car cela nécessite la connexion des utilisateurs dans le programme principal afin de différencier les différents comportements du programme selon leurs droits.

Concernant la gestion de l'espace mémoire, lorsque l'utilisateur crée un fichier dont la taille est plus grande que les blocs de mémoires restants, une exception est levée et un message est affiché afin d'informer l'utilisateur qu'il n'y a plus d'espace disponible dans le répertoire.

Enfin, nous avons imposé la contrainte d'utiliser uniquement le caractère "/" dans les chemins, c'est-à-dire que "\" n'est pas considéré comme un séparateur valide.

Principaux algorithmes et types de données

Raffinage du programme principal

R0 : Afficher un menu permettant à l'utilisateur de gérer ses fichiers
R1 : Comment "Afficher un menu permettant à l'utilisateur de gérer ses fichiers"?
Initialiser (SGF);
Répéter
    Afficher les options des interfaces disponibles;
    Lire(choix);
    Démarrer l'interface choisie par l'utilisateur;
Jusqu'à choix = 99;
R2 : Comment "Afficher les options des interfaces disponibles"?
Ecrire("Available interfaces :");
Ecrire("1 - Terminal");
Ecrire("2 - Menu");
Ecrire ("99 - Exit");
Ecrire ("Please enter the number to the corresponding interface : ");
R2 : Comment "Démarrer l'interface choisie par l'utilisateur"?
Selon choix dans
    1 => Démarrer l'interface terminal;
    2 => Démarrer l'interface menu;
    99 => null;
    Autres => Lever l'exception Choix_Invalide;
Fin Selon;
R3 : Comment "Démarrer l'interface terminal"?
Afficher les description des commandes disponibles;
Afficher (">");
Lire (input);
Tant que input /= "exit" faire
    Récupérer la commande, les arguments et ses paramètres;
    Traiter chaque commande saisie par l'utilisateur;
    Afficher (">");
    Lire(input);
Fin tant que;
Afficher("Exiting terminal");
R4: Comment "Récupérer la commande, les arguments et ses paramètres" ?
vecteur.clear;
Start <-- 0;
Pour i de 1 à Taille(input) faire
    Si input(i) = ' ' alors
        vecteur.append(input(Start .. i-1));
    Fin Si;
    Start <-- i + 1;
Fin pour;
Récupérer le dernier mot;
R5 : Comment "Récupérer le dernier mot" ?

```
        Si Start <= Taille(input) alors
                Ajouter input(Start..Taille(input)) dans le vecteur);
        Fin si;
R4 :    Comment “Traiter chaque commande saisie par l’utilisateur” ?
        cmd <-- vecteur(1);
        Si cmd = “pwd” alors
                Vérifier le nombre d’arguments;
                Afficher (Get_Current_Directory(sgf));
        SinonSi cmd = “touch” alors
                Vérifier le nombre d’arguments;
                Create_File(sgf,vecteur(1),vecteur(2));
        SinonSi cmd = “size” alors
                Vérifier le nombre d’arguments;
                Change_File_Size(Sgf,vecteur(1),vecteur(2));
        SinonSi cmd = “mkdir” alors
                Vérifier le nombre d’arguments;
                Create_Directory(Sgf,vecteur(1));
        SinonSi cmd = “cd” alors
                Vérifier le nombre d’arguments;
                Si vecteur.last_index = 1 alors
                        Current_Directory(Sgf,vecteur(1));
                Sinon
                        Current_Directory(Sgf);
                Fin si;
        SinonSi cmd = “ls” alors
                Vérifier le nombre d’arguments;
                Afficher le contenu du répertoire;
        SinonSi cmd = “rm” alors
                Vérifier le nombre d’arguments;
                Supprimer le fichier ou le répertoire donné;
        SinonSi cmd = “mv” alors
                Vérifier le nombre d’arguments;
                Move(Sgf,vecteur(1),vecteur(2));
        SinonSi cmd = “cp” alors
                Si vecteur.last_index = 3 et alors vecteur(1) = ”-r” alors
                        Copy_Recursive (Sgf,vecteur(2),vecteur(3));
                Sinon
                        Vérifier le nombre d’arguments;
                        Copy(Sgf,vecteur(1),vecteur(2));
                Fin Si;
        Sinon
                Afficher (“The command “ & input & “ was not found”);
        Fin Si;
R5 :    Comment “Vérifier le nombre d’argument” ?
        Si nb_argument > max_nb alors
                Lever l’exception Too_Many_Argument;
        SinonSi nb_argument < min_nb alors
                Lever l’exception Not_Enough_Argument;
```

```
        Fin Si;
R5 :    Comment “Afficher le contenu du répertoire”?
        Si vecteur.last_index = 2 et alors vecteur(1) = “-rl” ou vecteur(1) = “-lr” alors
                Afficher(List_Files_Recursive(Sgf,vecteur(2), True));
        SinonSi vecteur.last_index = 1 et alors vecteur(1) = “-rl” ou vecteur(1) = “-lr” alors
                Afficher(List_Files_Recursive(Sgf, True));
        SinonSi vecteur.last_index = 2 et alors vecteur(1) = “-r” alors
                Afficher(List_Files_Recursive(Sgf,vecteur(2)));
        SinonSi vecteur.last_index = 1 et alors vecteur(1) = “-r” alors
                Afficher(List_Files_Recursive(Sgf));
        SinonSi vecteur.last_index = 2 et alors vecteur(1) = “-l” alors
                Afficher(List_Files(Sgf,vecteur(2),True));
        SinonSi vecteur.last_index = 1 et alors vecteur(1) = “-l” alors
                Afficher(List_Files(Sgf,vecteur(2),True));
        SinonSi vecteur.last_index = 1 alors
                Afficher(List_Files(Sgf,True));
        Sinon
                Afficher(List_Files(Sgf));
        Fin Si;

R3 :    Comment “Démarrer l’interface menu” ?
        Répéter
                Afficher les options de commandes;
                Afficher(“Command number :”);
                Lire(choix);
                Selon choix dans
                        1 => Print_Current_Working_Directory(Sgf);
                        2 => Add_New_File(Sgf);
                        3 => Add_New_Directory(Sgf);
                        4 => Change_File_Size(Sgf);
                        5 => Change_Current_Directory(Sgf);
                        6 => Print_Directory_Content(Sgf,False);
                        7 => Print_Directory_Content(Sgf,True);
                        8 => Remove_File_Or_Directory(Sgf,False);
                        9 => Remove_File_Or_Directory(Sgf,True);
                        10 => Move_Or_Rename(Sgf);
                        11 => Copy_File(Sgf);
                        12 => Copy_Directory(Sgf);
                        13 => Archive_Directory(Sgf);
                        99 => null;
                        others => lever l’exception Invalid_Choice;
                FinSelon;
        Jusqu’à choix = 99;
```

Types de données

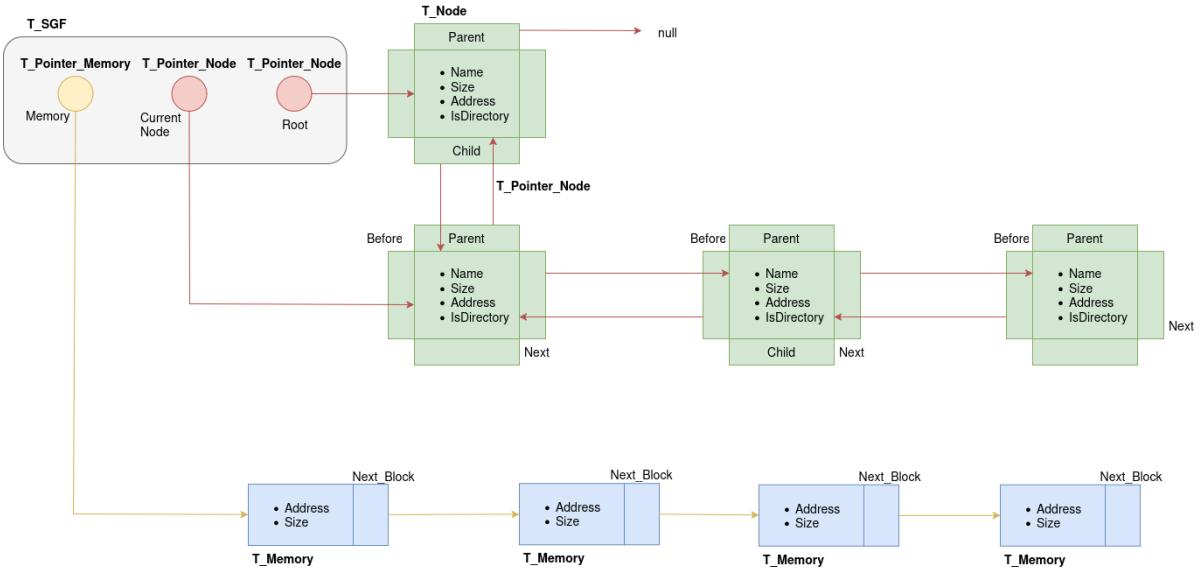

Figure 2 : Représentation visuelle des types de données

Nous avons créé les structures de données suivantes pour représenter la hiérarchie du SGF ainsi que l'allocation de mémoire pour les fichiers et répertoires créés dans le système. Cela facilite également le stockage et la manipulation des données dans le SGF.

Structure de données	Description
T_Node	Un nœud du type enregistrement dans une structure arborescente. Chaque nœud représente soit un fichier, soit un répertoire selon la valeur de la variable booléenne isDirectory. Un nœud possède un nom, une taille et une adresse mémoire. Il contient également plusieurs pointeurs de type T_Pointer_Node permettant de référencer ses nœuds enfants, son nœud parent ainsi que ses nœuds voisins, ce qui permet de construire et de parcourir la structure d'arbre du SGF.
T_Pointer_Node	Un type pointeur qui référence un objet de type T Node. Il est utilisé dans deux attributs de la structure T_SGF, l'un pour pointer vers la racine de l'arbre du SGF et l'autre pour pointer vers le nœud courant, correspondant au répertoire dans lequel se situe l'utilisateur. Les nœuds de type T_Node utilisent également T_Pointer_Node pour se relier entre eux et former la structure hiérarchique des fichiers et répertoires.
T_Memory	T_Memory est une cellule du type enregistrement dans une liste chaînée simple. Elle permet de représenter les blocs d'espace mémoire vide disponible pour l'allocation aux nouveaux fichiers et répertoires. Chaque cellule T_Memory contient l'adresse de début d'un bloc de mémoire vide, la taille de ce bloc ainsi qu'un pointeur de type T_Pointer_Mémoire vers la cellule suivante.

T_Pointer_Memory	Le type T_Pointer_Memory est un type pointeur qui référence un objet de type T_Memory. Il est utilisé dans la structure T_SGF afin de pointer vers la première cellule de la liste chaînée représentant l'allocation de l'espace mémoire du SGF.
T_SGF	Le type T_SGF est un type d'enregistrement qui regroupe les informations pertinentes à la manipulation du SGF. Il contient le pointeur vers la racine du SGF, le pointeur vers le répertoire courant de l'utilisateur ainsi que le pointeur vers la liste chaînée représentant l'espace mémoire du SGF.

Fonctionnement de gestion de mémoire

La structure liste chaînée utilisée pour la gestion de mémoire représentent uniquement les blocs de mémoire libre. Cette représentation est plus efficace que représenter les blocs occupés car cela réduit le nombre de cellules à gérer. En plus, cela simplifie l'opération d'allocation car lorsqu'un un nouveau fichier ou répertoire doit être alloué, il suffit de parcourir la liste des blocs libres pour trouver un bloc de taille suffisante.

Allocation de mémoire

R0 : Allouer de la mémoire lors de la création d'un fichier ou d'un répertoire
Récupérer le premier bloc de mémoire libre;
Parcourir la liste chaînée pour identifier un bloc de mémoire suffisamment grand;
Vérifier si la fin de liste est atteinte sans avoir trouvé un bloc libre;
Allouer le bloc de mémoire

R1 : Comment "Récupérer le premier bloc de mémoire libre" ?
temp_bloc <-- Sgf.Memory;

R1 : Comment "Parcourir la liste chaînée pour identifier un bloc de mémoire suffisamment grand" ?
Tant que temp_block /= null et alors temp_block^.size < size faire
    prev_block <-- temp_block;
    temp_block <-- temp_block^.next_block
Fin tant que;

R1 : Comment "Vérifier si la fin de liste est atteinte sans avoir trouvé un bloc libre" ?
Si temp_bloc = null alors
    Lever l'exception Size_Limit_Reach;
Fin si;

R1 : Comment "Allouer le bloc de mémoire" ?
Si temp_block^.size = size et alors temp_block /= null alors
    Supprimer temp_block de la liste;
SinonSi
    temp_block^.size <-- temp_block.all.size - size;
FinSi;

R2 : Comment "Supprimer temp_block de la liste"?
prev_bloc^.Next_Block <-- temp_block^.Next_Block;
Libérer(temp_block);

Désallocation de mémoire

R0 : “Désallouer de la mémoire lors de la suppression d’un fichier ou d’un répertoire”

R1 : Comment “désallouer de la mémoire lors de la suppression d’un fichier ou d’un répertoire”?
Récupérer le bloc de mémoire le plus proche;
Répartir la mémoire;
Fusionner les blocs de mémoire;

R2 : Comment “Récupérer le bloc de mémoire le plus proche”?
Tant que temp_block^.next_block /= Null et alors temp_block^.next_block^.Address < node^.Address faire
    temp_block <-- temp_block^.next_block;
Fin tant que;

R2 : Comment “Répartir la mémoire”?
Si temp_block^.Address + temp_block^.Size = node^.Address faire
    Ajouter la mémoire sur le bloc courant;
Sinon
    Créer un nouveau bloc de mémoire;
Fin si;

R2 : Comment “Fusionner les blocs de mémoire”?
Si temp_block^.Next_Block /= Null and then temp_block^.Address + temp_block^.Size = temp_block^.Next_Block^.Address faire
    temp_block^.Size <-- temp_block^.Size + temp_block^.Next_Block^.Size;
    remove_block <-- temp_block^.Next_Block;
    temp_block^.Next_Block <-- temp_block^.Next_Block^.Next_Block;
    Libérer(remove_block);
Fin si;

R3 : Comment “Ajouter la mémoire sur le bloc courant”?
temp_block^.Size <-- temp_block^.Size + node^.Size;

R3 : Comment “Créer un nouveau bloc de mémoire”?
temp_block.Next_Block <-- new T_Memory'(node^.Address, node^.Size, temp_block^.Next_Block);
temp_block<-- temp_block^.Next_Block;

Méthodologie de tests

Pendant la phase de développement, nous avons créé des tests unitaires afin de vérifier le bon fonctionnement des procédures et fonctions dans un module. Nous avons également testé la gestion des exceptions dans les tests unitaires afin de vérifier que le programme réagit correctement aux situations d'erreur. (cf. Annexe - Figure A1 et Annexe - Figure A2)

À la fin du développement, nous avons également réalisé un scénario de test, y compris des actions suivantes: (cf Annexe - Figure A3)

1. Création des répertoires
2. Création des fichiers
3. Supprimer des fichiers
4. Déplacer un fichier
5. Copier un fichier
6. Archiver un répertoire

Difficultées rencontrées et solutions adoptées

Nous avons des difficultés au début, pendant la phase de conception, car nous n'avons pas une idée claire de structure du SGF, ni de sa représentation éventuelle en utilisant des structures de données en Ada. Étant donné que nous avions davantage d'expériences avec les langages orientés objet, nous avons d'abord conçu la structure de SGF en suivant le design pattern Composite. Après en avoir discuté avec les professeurs, nous avons remarqué que les structures de données en Ada sont différentes de celles utilisées en programmation orientée objet. Nous avons donc adapté notre conception et adopté une structure plus appropriée aux spécificités du langage Ada.

Nous avons également la difficulté de débogage car il y a certains scénarios que nous n'avions pas pris en compte pendant le développement du programme. Le message d'erreur rencontré n'était pas très parlant sur l'origine du problème car les exceptions à ce stade de développement n'étaient pas assez robustes. Donc, nous avons décidé d'utiliser le débogueur fourni par GNAT Studio afin d'identifier l'origine du problème. Grâce à cet outil, nous avons trouvé les problèmes facilement et efficacement.

Nous avons également mal compris, dans un premier temps, le fonctionnement de la gestion de la mémoire, en mettant en place une "gestion de mémoire" simple reposant sur une taille maximale de stockage qui diminuait à chaque création de fichier ou de répertoire.

Cependant, après avoir vérifié notre approche avec les professeurs, nous avons réalisé que cette manière de gérer la mémoire n'était pas correcte. Une fois le fonctionnement approprié compris, nous avons réussi à mettre en œuvre une version similaire à celle de Valgrind, mais sans le processus de défragmentation dans le cas où tous les blocs libres ne sont pas suffisamment grands.

Organisation du groupe

Nous avons commencé par travailler ensemble durant la phase de conception afin de nous mettre d'accord sur les structures de données à manipuler. Ensuite, nous avons identifié et réparti les fonctionnalités attendues à partir du cahier des charges entre nous. Voici la répartition des fonctionnalités à développer :

Ellisa	● Création de SGF qui ne contient que du dossier racine ● Obtention du répertoire de travail ou répertoire courant ● Création d'un fichier ● Modifier la taille d'un fichier ● Création d'un répertoire et d'un fichier ● Affichage des fichiers et des répertoires ● Interface menu
Tristan	● Changement du répertoire courant ● Affichage des fichiers et des répertoires ● Suppression d'un fichier ou d'un répertoire ● Déplacement d'un fichier ● Copie d'un fichier ou d'un répertoire ● Gestion de mémoire ● Interface terminal

Nous avons travaillé de manière autonome tout en restant toujours disponibles pour utiliser la méthode du pair programming, issue de la méthodologie Agile, en cas de problème dans le code.

Même si nous ne suivions pas forcément la méthodologie Scrum de manière stricte, nous avons tout de même mis en place une version simple du Daily Scrum. À chaque séance de travail ensemble, nous nous assurions de fusionner nos deux branches de code sur Git et de discuter des problèmes rencontrés ainsi que des tâches à réaliser pour la journée. Cela nous a permis d'assurer la compatibilité de nos codes et de résoudre les problèmes de manière efficace.

Afin de gérer les différentes versions du code entre nous pendant la phase de développement, nous avons utilisé Git avec l'outil Github Desktop, qui fournit une interface pour effectuer la gestion des versions.

Bilan technique

État d'avancement du projet

À l'instant, le projet comporte toutes les fonctionnalités demandées par le client, incluant :

- Chaque commande, dont :
    - La création d'un SGF,
    - L'obtention du répertoire de travail,
    - La création d'un fichier,
    - La modification de la taille d'un fichier,
    - La création d'un répertoire,
    - Le changement de répertoire courant,
    - L'affichage du contenu du répertoire sélectionné,
    - L'affichage récursive du contenu du répertoire sélectionné,
    - La suppression d'un fichier,
    - La suppression d'un répertoire,
    - Le déplacement ainsi que le renommage d'un fichier,
    - Le copiage d'un fichier,
    - Le copiage d'un répertoire.
- Les interfaces :
    - Mode Terminal pour ceux qui connaît mieux l'utilisation des commandes pour manipuler les fichiers et les répertoires dans un SGF
    - Mode Menu pour faciliter la prise en main par des utilisateurs qui ne sont pas familiers avec les commandes du terminal.
- La gestion de mémoire.

Perspectives d'amélioration

En revanche, certains programmes pourraient être rajoutés pour améliorer son utilisation, comme l'addition de la gestion des droits d'utilisateur pour permettre l'accès au dossier ou l'exécution de fichiers pour des utilisateurs spécifiques, et donc rajouter aussi une fenêtre de connexion.

En plus, en raison du manque du temps, nous n'avons pas eu l'opportunité de développer l'algorithme de défragmentation pour la gestion de la mémoire. Donc, une version future du SGF intégrant le processus de défragmentation pourrait être envisagée afin d'éviter le gaspillage de l'espace mémoire.

Bilan personnel et individuel

Temps passé

	Tristan Gallardo		Ellisa Ee	
	Temps passé en cours	Temps passé en dehors des cours	Temps passé en cours	Temps passé en dehors des cours
Conception	3h30	0h	3h30	0h
Implantation	21h45	3h30	18h50	4h00
Mise au point	1h45	1h45	5h15	0h
Rapport	4h30	0h	4h15	2h15
Total	31h30	5h15	31h30	6h15

Enseignements tirés de ce projet

Tristan Gallardo

Grâce à ce projet, je comprends maintenant mieux le fonctionnement des fichiers sur Windows et Linux, comme la manière dont ils sont créés, modifiés, ou même, comment ils sont structurés entre eux.

Cela m’a surtout entraîné sur les pointeurs, commettant de moins en moins d'erreurs sur ce sujet, et améliorant ainsi ma manière de coder avec tels pointeurs.

Enfin, ce sujet en général m’a permis d’obtenir de meilleures connaissances en Ada, découvrant des structures de contrôle qui m'étaient encore inconnues auparavant grâce aux recherches effectuées, et d’optimiser/de refactorer plus facilement mon code.

Ellisa Ee

Au cours de ce projet, j’ai appris plus en profondeur le fonctionnement d’un SGF ainsi que les principes d’allocation de la mémoire. Ces notions m’ont permis de mieux comprendre comment les données sont stockées, organisées et manipulées en mémoire, ce qui constitue une base importante pour le cours de systèmes d’exploitation à venir.

Grâce aux cours de programmation suivis à l’IUT Informatique, je disposais déjà d’une base de connaissance en langage Ada. Cependant, ce projet m’a donné l’opportunité d’approfondir mes connaissances, en particulier sur la manipulation des pointeurs et l'utilisation de structures de données plus complexes.

Enfin, la durée allouée à ce projet était relativement courte comparée à celle des projets de programmation en Ada réalisés précédemment. Cette contrainte de temps m’a poussé à

mieux m'organiser, à planifier les différentes étapes du développement et à prioriser les tâches essentielles. J'ai ainsi amélioré la gestion du temps et ma capacité à travailler sous pression.

Annexe

```
procedure Get_Current_Working_Directory_Test (Sgf : out T_SGF) is

begin
    Construct_SGF_Example(Sgf);
    pragma Assert (Get_Current_Directory(Sgf)="/");
    Current_Directory(Sgf,"/home");
    pragma Assert (Get_Current_Directory(Sgf)="/home/");
    Current_Directory(Sgf,"./user1");
    pragma Assert (Get_Current_Directory(Sgf)="/home/user1/");
    Current_Directory(Sgf,"./pim");
    pragma Assert (Get_Current_Directory(Sgf)="/home/user1/pim/");
    Current_Directory(Sgf,"./projet");
    pragma Assert (Get_Current_Directory(Sgf)="/home/user1/pim/projet/");
end Get_Current_Working_Directory_Test;
```

Figure A1 : Exemple d'un test unitaire pour la fonction get_current_directory(Sgf)

```
procedure Create_Directory_Exception_Test(Sgf : out T_SGF) is
    Directory_Name_Conflict : Boolean := false;
    Name_Is_A_Dot : Boolean := false;
    Name_Is_Two_Dot : Boolean := false;
begin
    -- Create a new directory with a name that already exists in the target
    begin
        Create_Directory(Sgf,"relative-path");
    exception
        when Directory_Exists_Error => Directory_Name_Conflict := True;
    end;
    -- Create directory with a "." as name
    begin
        Create_Directory(Sgf,".");
    exception
        when Dot_Name_Error => Name_Is_A_Dot := True;
    end;
    -- Create directory with a ".." as name
    begin
        Create_Directory(Sgf,"..");
    exception
        when Dot_Name_Error => Name_Is_Two_Dot := True;
    end;

    pragma Assert(Directory_Name_Conflict);
    pragma Assert(Name_Is_A_Dot);
    pragma Assert(Name_Is_Two_Dot);

end Create_Directory_Exception_Test;
```

Figure A2 : Exemple d'un test unitaire pour la gestion des exception

```
procedure General_Scenario_Test (Sgf :out T_SGF) is
begin
    Construct_SGF_Example (Sgf);
    Remove(Sgf, "/home/user1/pim/tp/tp1/newton.adb");
    begin
        Remove(Sgf, "/home/user1/pim/tp/tp1/newton.adb");
        pragma Assert(False);
    exception
        when Dir_Not_Found => pragma Assert(True);
    end;
    Move(Sgf, "/home/user1/pim/tp/tp1/min_max_serie.adb", "/usr/local/share/new_name");
    pragma Assert(Get_Name(Sgf, "/usr/local/share/new_name", false) = "new_name");
    Copy(Sgf, "/home/user1/pim/tp/tp1/min_max_serie.py", "/usr/local/share");
    pragma Assert(Get_Name(Sgf, "/usr/local/share/min_max_serie.py", false) = "min_max_serie.py");
    Archive_Directory(Sgf, "/home/user1/pim/projet/home.tar","/home");
    pragma Assert(Get_Size(Sgf,"/home/user1/pim/projet/home.tar",False)=10030);
end General_Scenario_Test;
```

Figure A3 : Exemple du test du test scénario général