Méthodologie de la Programmation.
Projet 2025 - 2026

Un Système de Gestion de Fichiers

Le composant logiciel du système d'exploitation chargé de la gestion des fichiers et des répertoires d'un ordinateur est appelé **Système de Gestion de Fichiers** ou (**SGF**). De manière générale, ce composant permet de manipuler des fichiers, des répertoires et de gérer l'espace mémoire occupée par ces fichiers.

Dans le cadre de ce projet, nous nous proposons de réaliser un système de gestion de fichiers.

Fichiers et répertoires

Les SGF organisent les répertoires de manière hiérarchique avec un répertoire racine souvent appelé **root** et représenté par « / ».

Chaque répertoire est défini par un nom, des droits d'accès, un répertoire parent [1] et un contenu constitué d'autres répertoires et fichiers. Un fichier est caractérisé par un nom, une taille [2], des droits d'accès et plusieurs autres informations.

Voici un exemple d'arborescence de répertoires et fichiers. Seuls quelques fichiers des répertoires 'projet' et 'tp1' sont montrés. Par exemple le fichier 'exemple.adb' est un fichier du répertoire 'projet' dont le répertoire parent est 'pim'.

```
/
 \- home
 |   \- user1
 |          pim
 |           \- projet
 |           |   |- exemple.adb
 |           \- tp
 |               \- tp1
 |                   |- min_max_serie.adb
 |                   |- min_max_serie.py
 |                   |- newton.adb
 |                   |- puissance.adb
 \- usr
     \- local
         \- share
```

Pour manipuler fichiers et répertoires, les SGF offrent à l'utilisateur une interface de manipulation qui permet, entre autres, de créer, supprimer, renommer, copier, consulter, crypter, etc. des fichiers et des répertoires. Le SGF se charge de libérer ou d'allouer les espaces mémoire sur le disque de façon complètement transparente pour l'utilisateur.

1. Seul le répertoire racine n'a pas de répertoire parent.
2. On suppose la taille exprimée en nombre de caractères et on se limite ici à des fichiers texte.

Cette interface utilise la notion de répertoire courant, notée « . » (`pwd` pour « print working directory »), pour désigner le répertoire de travail (en cours de manipulation). Par défaut, les commandes sont relatives à ce répertoire de travail (par exemple la commande `ls`). A partir du répertoire de travail il est donc possible, grâce à des opérations de parcours offertes par le SGF d'accéder à un fichier ou à un autre répertoire. Un fichier ou un répertoire peut être désigné grâce à un chemin qui peut être absolu ou relatif au répertoire courant.

Voici quelques exemples de chemins absolus (on donne tous les noms de répertoires depuis la racine pour arriver jusqu'au répertoire ou fichier considéré) :

```
/
/home
/usr/local/share
/home/user1/pim/projet/exemple.adb
```

Voici quelques exemples de chemins relatifs (on suppose que le répertoire courant est `/home/user1/pim/tps`) :

```
.                        -- le répertoire courant
..                       -- le répertoire parent
../projet/exemple.adb    -- le fichier exemple.adb du répertoire projet
                         -- du répertoire parent du répertoire courant
```

Partie I

Interface de manipulation d'un système de gestion de fichiers

Pour créer, manipuler et supprimer des fichiers et des répertoires, le SGF offre plusieurs commandes ou opérations. Ces commandes sont utilisées par des programmes qui créent ces fichiers : un éditeur de texte qui permet à l'utilisateur de créer ou modifier des fichiers, un compilateur qui engendre du code dans un fichier à partir d'une code source lui-même dans un autre fichier, une base de données qui stocke dans des fichiers ses tables, etc.

Parmi ces commandes ou opérations, on peut lister :

- création d'un SGF ne contenant que le dossier racine (elle correspond au formatage d'un disque par exemple),
- obtention du répertoire de travail ou répertoire courant (commande `pwd`),
- création d'un fichier,
- modifier la taille d'un fichier (c'est le résultat de l'utilisation d'un éditeur de texte),
- création d'un répertoire,
- changement du répertoire courant en précisant le chemin du nouveau répertoire courant (commande `cd`),
- affichage du contenu, fichiers et répertoires, du répertoire désigné par un chemin (exemples `ls ../projet` ou `ls /home/user1/pim/projet/`). Si le chemin n'est pas précisé, le contenu du répertoire courant est affiché (commande `ls`),
- affichage des fichiers et des répertoires du répertoire courant et de tous les fichiers et répertoires de tous les sous-répertoires (commande `ls -r`),
- suppression d'un fichier (commande `rm`),
- suppression d'un répertoire qu'il soit vide ou non (commande `rm -r`),
- déplacement (et renommage éventuel) d'un fichier (commande `mv`),
- copie d'un fichier ou d'un répertoire (commande `cp -r`),

— archiver un répertoire (commande `tar`) qui permet de produire, à partir d'un répertoire, un unique fichier dont la taille est la somme des tailles des fichiers contenus dans ce répertoire et ses sous-répertoires.
Remarque. La commande `tar` ne compresse pas le contenu des fichiers.

> La liste ci-dessus n'est pas exhaustive, il existe plusieurs autres opérations qui permettent de manipuler fichiers et répertoires. En plus de celles qui sont demandées, **vous pouvez développer d'autres fonctionnalités/commandes à votre convenance.**

Remarques.
— Toutes les commandes du SGF manipulent un (ou des) chemin(s). Il est important de traiter la notion de chemin relatif et absolu dans vos programmes.
— Les commandes d'un SGF manipulent des noms de fichiers ou de répertoires à l'aide d'expressions régulières. Par exemple `*.adb`, `projet*pim.adb`, `p?m.adb` etc. Il est demandé de prendre en compte les noms de fichiers et de répertoires qui utilisent les opérateurs `*` et `?`
— La plupart des commandes peuvent être construites à partir de commandes de base ou d'autres commandes.
— Il faudra éviter les codes redondants.
— Vous pouvez développer des commandes supplémentaires à votre convenance.

Travail demandé.

L'objectif principal du projet est de :

1. **simuler un SGF.** On définira un module SGF qui devra obligatoirement s'appuyer sur un module qui spécifie et implante une structure de donnée de type *Arbre*.
2. tester votre programme
3. proposer un mini-interpréteur de commande qui donne accès aux commandes énumérées en section . On décrira deux interfaces d'accès,
   (a) l'une utilisant un menu (choix d'une commande parmi une liste de commandes)
   (b) et la seconde utilisant une ligne de commande (commande suivie des paramètres séparés par des espaces comme dans un terminal :

```
> ls
> touch readme.txt
> mkdir projet
> ls
projet readme.txt
> _
```

Partie II

Gestion de la mémoire

A ce stade du projet, nous n'avons pas abordé la gestion de la mémoire occupée par les différents fichiers et répertoires.

Pour la suite, on suppose que l'espace mémoire sur le disque est de 1 To (1 Terra Octet).

Les informations associées à chaque répertoire et fichier sont complétées par une adresse (sur le disque) de début (base) et sa taille.

Chaque répertoire aura une taille fixe de 10 ko. On ne peut pas créer un nouveau répertoire ou fichier s'il n'y a pas assez de mémoire *contigue* disponible.

Les opérations de création, de destruction et de copie de fichiers et de compactage de l'espace mémoire libre ont un effet sur la gestion de l'espace mémoire sur le disque.

Travail demandé.

5 Parmi les opérations identifiées en section , donner celles qui ont un effet sur l'espace mémoire sur le disque. Justifier vos réponses.

6 Compléter les programmes précédents pour toutes les commandes qui ont un effet sur la gestion de l'espace mémoire sur le disque. Pour cela, il est recommandé de définir une structure de données permettant de représenter les espaces mémoire libres sur le disque. Notons que la signature de chaque sous-programmes associé à chacune de ces commandes ne doit pas être modifiée. Seules spécification et implantation devront être modifiées pour prendre en compte la gestion de la mémoire.

1 Exigences pour la réalisation du projet

1. Le projet se déroulera en binôme. Vous devez indiquer aux enseignants **les noms et prénoms de votre binôme pour la séance de projet du Mardi 06 Janvier 2026 à 16h15.**
2. Le langage utilisé est le langage Ada limité aux concepts vus en cours, TD et TP.
3. Le programme devra compiler et fonctionner pour être démontré aux enseignants.
4. L'ensemble des concepts du cours devront être mis en œuvre, en particulier :
   - Ecriture des spécifications pour tous les programmes et sous-programmes
   - Conception en utilisant la méthode des raffinages
   - Justification des choix des types de données manipulés
   - Conception de modules : encapsulation, généricité, TAD, etc.
   - Définition des tests et le processus de test mis en place
5. Les développements associés au projet comprendront les répertoires suivants.
   - Le répertoire « **livrables** » ne devra contenir que les livrables explicitement demandés.
   - Le répertoire « **src** » contiendra les sources de votre application.
   - Le répertoire « **doc** » sera utilisé pour le rapport.

   Vous pouvez bien sûr créer d'autres répertoires si vous en avez besoin.

2 Livrables

Les **documents à rendre** sont :
- la **spécification des modules** en Ada (et donc la définition des principaux types dans la partie privée) (extension `.ads`),
- les **programmes de test** (`test_*.adb`),
- une archive contenant le **code source final** de vos programmes.
- un **rapport** (rapport.pdf) contenant au moins :
  - un résumé qui décrit l'objectif et le contenu du rapport (10 lignes maxi),

- une introduction qui présente le problème traité [3] et le plan du document
- l'architecture de l'application en modules
- la présentation des principaux choix réalisés
- la présentation des principaux algorithmes et types de données
- la démarche adoptée pour tester le programme
- les difficultés rencontrées et les solutions adoptées en justifiant vos choix (en particulier quand vous avez envisagé plusieurs solutions)
- l'organisation du groupe (qui a fait quoi, etc.)
- un bilan technique donnant un état d'avancement du projet et les perspectives d'amélioration / évolution
- une bilan *personnel* et *individuel* : intérêt, temps passé, temps passé à la conception, temps passé à l'implantation, temps passé à la mise au point, temps passé sur le rapport, enseignements tirés de ce projet, etc.

— Un **manuel utilisateur** (manuel.pdf) qui décrit comment utiliser les programmes développés et qui est illustré avec des copies d'écran.

3 Principaux critères de notation

Voici les principaux critères qui seront pris en compte lors de la correction du projet :

— le respect du cahier des charges,
— la qualité des raffinages,
— la facilité à comprendre la solution proposée,
— la pertinence des modules définis,
— la pertinence des sous-programmes définis,
— la qualité des sous-programmes : ils ne doivent pas être trop longs, ne pas faire trop de choses, ne pas avoir trop de structures de contrôle imbriquées. Dans ces cas, il faut envisager de découper le sous-programme !
— la bonne utilisation de la programmation par contrat (en particulier les préconditions et les invariants de type) et de la programmation défensive,
— la pertinence des tests réalisés sur les sous-programmes
— la pertinence des types utilisateurs définis,
— l'absence de code redondant,
— le choix des identifiants,
— les commentaires issus des raffinages,
— la bonne utilisation des commentaires,
— le respect de la solution algorithmique (les raffinages) dans le programme,
— la présentation du code (le programme doit être facile à lire et à comprendre),
— l'utilisation des structures de contrôle adéquates,
— la validité du programme,
— la robustesse du programme,
— la bonne utilisation de la mémoire dynamique (valgrind).

4 Date de restitution

Le projet devra être rendu le **30 Janvier 2026** à **23h00** par **Moodle**.

3. La présentation du problème doit être concise car les enseignants connaissent le sujet !