TP à rendre 1

Dans cet exercice vous devez écrire le programme `backup` dont la syntaxe est :

```
./backup source reference destination
```

qui effectue une sauvegarde « incrémentale » du répertoire `source` dans le répertoire `destination` en se basant sur le répertoire `reference`. Le répertoire `source` *doit* exister. Le répertoire `reference` *peut* exister. Le répertoire `destination` *ne doit pas* exister. Tous les arguments peuvent être des chemins absolus ou relatifs.

L'idée est que le répertoire `reference` est la *précédente* sauvegarde de `source`. La sauvegarde est incrémentale parce qu'on ne va utiliser de l'espace disque que pour les fichiers qui ont été modifiés depuis cette précédente sauvegarde. En supposant qu'on a un répertoire `../../progsys`, on fait une première sauvegarde complète un dimanche, en utilisant la commande `cp` (*avec* l'option `-a`) :

```
cp -ar ../../progsys /data/dimanche
```

(ou, mieux, on utilise le programme `backup` avec un répertoire de référence inexistant.) On peut alors faire une sauvegarde incrémentale lundi soir (la référence est la sauvegarde de dimanche) :

```
./backup ../../progsys /data/dimanche /data/lundi
```

Puis, chaque jour de la semaine, on fait une nouvelle sauvegarde incrémentale par rapport à celle de la veille. Par exemple mardi soir :

```
./backup ../../progsys /data/lundi /data/mardi
```

Mercredi aussi on travaille, puis :

```
./backup ../../progsys /data/mardi /data/mercredi
```

Et ainsi de suite. On souhaite que chaque nouvelle sauvegarde utilise un espace disque proportionnel à la taille des fichiers *modifiés*, et ne pas utiliser plusieurs fois l'espace disque nécessaire aux fichiers inchangés.

On désigne par `source/chemin` le chemin d'accès à un fichier à l'intérieur du répertoire `source` ou d'un de ses sous-répertoires, à n'importe quelle profondeur (par exemple `source/L2/progsys/tp3/backup.c`). Le fonctionnement du programme est le suivant :

- Seuls les fichiers ordinaires, les répertoires et les liens symboliques de `source` sont sauvegardés. Les autres types de fichiers sont ignorés (mais un message sur la sortie standard doit indiquer le chemin de chaque fichier ignoré).
- Un fichier ordinaire `source/chemin` dont il existe une version *inchangée* dans `reference/chemin` n'est pas copié, mais un lien *physique* nommé `destination/chemin` est créé, et ce lien doit désigner `reference/chemin`.
- Un fichier ordinaire `source/chemin` doit être copié dans `destination/chemin` si il n'existe pas de fichier `reference/chemin`, *ou* si ce dernier fichier existe mais que le fichier `source/chemin` a été modifié depuis la sauvegarde de `reference`.
- Lorsqu'un fichier ordinaire est copié dans `destination`, le programme doit attribuer à la copie les mêmes permissions et les mêmes dates de dernière modification/accès que le fichier d'origine.
- Tous les sous-répertoires de `source` sont toujours reproduits dans `destination` (il n'est pas possible de créer un lien physique vers un répertoire sous Unix).
- Un lien *symbolique* `source/chemin` est reproduit à l'identique (c'est-à-dire avec exactement la même cible) dans `destination/chemin`.

Dans cette description, on considère qu'un fichier est *inchangé* (par rapport à la sauvegarde de référence) si il a les mêmes dates de dernière modification/accès et les mêmes permissions. Du code pour le test des dates/permissions et pour la modification de ces dates/permissions vous est fourni dans le fichier `backup.c`, disponible sur Moodle.

L'avantage de ce système est que le répertoire `destination` est une copie complète et utilisable du répertoire `source`, et que deux sauvegardes successives ayant peu de fichiers différents utilisent une place raisonnable sur le disque. Ce mécanisme est utilisé par exemple dans l'outil `rsync` (avec l'option `--link-dest`), et dans l'outil *Time Machine* disponible sur les ordinateurs Apple (et sûrement dans d'autres logiciels).

On se donne une contrainte supplémentaire : un chemin d'accès à un fichier de doit jamais contenir plus de `CHEMIN_MAX` « caractères » (`char`). Vous devez définir cette constante dans votre programme par :

```
#ifndef CHEMIN_MAX
#define CHEMIN_MAX 1024 // par exemple
#endif
```

mais votre programme sera éventuellement recompilé avec une valeur prédéfinie. Si votre programme rencontre un chemin d'une longueur supérieure à `CHEMIN_MAX`, il doit s'arrêter avec un message d'erreur.

Programmation : Vous écrirez le programme `backup` en langage C en n'utilisant que des primitives systèmes. Vous pouvez utiliser des fonctions de bibliothèque pour la gestion des chaînes de caractères (`strcmp`, `snprintf`). Pour toute opération de manipulation de chaîne de caractères, vous devrez néanmoins garantir que votre programme ne va jamais écrire en dehors d'un tableau.

Votre programme doit compiler avec `cc -Wall -Wextra -Werror -Wvla` (utilisez le *Makefile* disponible sur Moodle). Les programmes qui ne compilent pas avec cette commande ne seront pas examinés. Votre programme ne devra pas comporter d'allocation dynamique de mémoire.

À rendre : Vous devrez rendre sur Moodle un *unique* fichier `backup.c` (Moodle sait qui vous êtes, il est inutile d'appeler votre programme `Jean-Claude_Dusse_L2S4_Printemps_2019-2020_backup.c`, et il est inutile aussi de rendre un fichier d'un autre nom ou une archive au format-du-jour – il y aura de sévères pénalités sinon).

Tests : Un script de test est mis à votre disposition sur Moodle. Celui-ci exécute votre programme sur des jeux de tests. Il vous affiche le résultat des 5 tests, qui servira de base à l'évaluation de votre rendu. Pour savoir comment l'utiliser, n'hésitez pas à lire ce script et le fichier de log généré : ils peuvent vous aider à mettre votre programme au point. N'hésitez pas non plus à contacter votre enseignant si vous constatez un comportement anormal ou si vous souhaitez ajouter un test.

Ce TP à rendre est individuel. On rappelle que la copie ou le plagiat sont sévèrement sanctionnés.