Dans ce TP vous allez réaliser une application de courriel sous la forme de scripts bash (vous avez peut-être commencé à écrire de tels scripts à la fin du TP1). Dans les 2 premiers exercices, ces scripts interagirons avec l'utilisateur sur le terminal. Dans le troisième exercice, nous utiliserons `kdialog` pour leur donner une interface graphique.

Exercice 1. scripts version "terminal" avec les données en dur.

Dans les scripts ci-dessous, vous utiliserez le programme `netcat` au-lieu de `telnet`. Ces programmes s'invoquent de la même façon et remplissent le même rôle, mais `telnet` est plutôt conçu pour une interaction sur un terminal, alors que `netcat` est plutôt conçu pour être utilisé dans des scripts. Comme `netcat` ne semble pas être installé par défaut, j'en ai placé une version compilée dans le répertoire `/pub/2A/ASR-Reseau`.

(a) écrivez et testez un script bash appelé `send.sh` pour vous envoyer un email sur le serveur `info2`.
(b) écrivez et testez un script bash appelé `fetch.sh` pour afficher le premier email de votre boite de couriel sur `info2` et le supprimer.

Dans ces scripts, vous mettrez directement toutes les données "en dur". Nous généraliserons par la suite.

Exercice 2. scripts version "terminal" avec bibliothèque de fonctions utilisant read.

Nous allons généraliser les scripts précédents : au lieu, de mettre tout "en dur", nous allons utiliser des fonctions qui vont demander à l'utilisateur de fournir les données nécessaires. Vous placerez toutes les fonctions qu'il vous faudra définir dans un fichier `lib.sh`. Les scripts feront un "import" de la bibliothèque `lib.sh` grâce à la commande "`.`" (un point). Autrement dit `send.sh` et `fetch.sh` devront commencer par la commande :

```
. lib.sh
```

(a) pour la fonction utilisée dans `send.sh` et qui demande le corps du message à l'utilisateur, on supposera dans un premier temps que le message n'a qu'une seule ligne.
(b) lorsque vous avez vérifié que tout marche, vous devrez maintenant modifier la fonction susmentionnée pour qu'elle permette des messages de plusieurs lignes.

Exercice 3. scripts version graphique avec bibliothèque de fonctions utilisant kdialog.

Lorsqu'un script a besoin d'interagir avec l'utilisateur, par exemple pour lui demander d'entrer un nom d'utilisateur, ou un mot de passe, c'est plus agréable de faire cela avec une interface graphique. C'est justement ce que permet `kdialog`. Dans cet exercice, vous allez écrire une nouvelle bibliothèque `lib2.sh` dans laquelle vous réimplémenterez toutes les fonctions de `lib.sh` mais en utilisant `kdialog` au lieu de `read`. La seule chose à modifier dans les scripts `send.sh` et `fetch.sh` est de remplacer la ligne :

```
. lib.sh
```

par

```
. lib2.sh
```

Voici quelques exemples d'utilisation de `kdialog` :

```
kdialog --inputbox "Nom"
```

Fait apparaitre une fenêtre graphique avec une boite d'édition dans laquelle l'utilisateur peut saisir du texte. Lorsque l'utilisateur clique OK, ou simplement appuie sur RETURN, `kdialog` écrit le texte saisi par ce dernier sur son `stdout`, puis s'arrete. On peut faire :

```
user=$(kdialog --inputbox "Nom")
```

pour demander son nom à l'utilisateur et l'affecter à la variable `user`. Notez, que puisque les fonctions de notre bibliothèque doivent écrire sur leur `stdout` les données qu'elles obtiennent de l'utilisateur, ce n'est pas nécessaire de mettre l'output de `kdialog` dans une variable pour en faire l'echo en suite : il suffit d'invoquer la commande `kdialog` et on aura directement le résultat sur le `stdout`. De manière similaire, on peut utiliser :

```
kdialog --password "Mot␣de␣passe"
```

pour obtenir un mot de passe, ou encore :

```
kdialog --textinputbox "Message" "" 80 50
```

pour obtenir le texte d'un message.

(a) expérimentez avec ces commandes pour vous familiariser avec leur usage
(b) essayez `kdialog --help` pour voir les autres commandes disponibles
(c) rédigez les fonctions de `lib2.sh` en utilisant `kdialog`