Outils système
TP 3

Tout ce TP doit être fait avec emacs.

Exercice 1 — Les bases

1. Faites un make générique pour java. Il s'utilisera de trois façon :
   — `make nomClass` et compilera avec javac la classe correspondante.
   — `make TestnomClass`. Il compilera avec javac la classe correspondante et la testera (compilera et lancera le fichier `test/TestnomClass.java`).
   — `make`. Il compilera et lancera le fichier principal `Main.java`.
   Vous pouvez tester votre `makefile` avec un de vos projets java (non fait avec IntelliJ) ou avec celui fourni sur Arche.

```
make arbre.class  // compiler le arbre.class a partir de arbre.java
make Testarbre    //Lancer le TestArbre.class (et compiler s'il faut)
make      //lancer le Main.class (et compiler s'il faut)
```

2. Prenez connaissance du projet C disponible sur Arche. Deux modules (affichage et jouer), un fichier main et un header seul.
   Les modules sont à compiler avec l'option `-c` de `gcc` (permettant de compiler des fichiers sans `main`).
   Le main est à compiler avec les fichiers objets des modules (`gcc main.c affichage.o jouer.o -o main`).
   `doxygen` est une commande permettant de créer une documentation. Il s'utilise sur le fichier executable final.
   Votre make doit :
   — Pouvoir compiler les modules (cela et éventuellement d'autres)
   — Compiler le main
   — créer la documentation
   — Nettoyer le dossier en enlevant tous les fichiers objets (pas le fichier exécutable du main).
   — Exporter l'exécutable et la bibliographie sous format zippé.
3. Faire un make pour les fichiers tex. Vous en trouverez un sur internet. Il génère plusieurs fichiers auxiliaires. Ajouter une commande à l'application de la règle utilisée pour générer `fichierTex.tex` qui supprime tous les fichiers `fichierTex.*` sauf le `.tex` et le `.pdf`.

Exercice 2 — Make automatique

Créez une commande `touchMake` qui attend deux arguments : `nomFichier`, `langage`. Elle crée un nouveau fichier `nomFichier`. Si l'argument `langage` est présent, elle ajoute une règle au `makefile`, en ayant préalablement listé les fichiers objets dont elle a besoin en cas de `c`, (les `#include "..."`). S'il n'y a pas de `makefile` elle doit le créer.

Vous pouvez aussi rajouter le texte de base du langage (la classe en java, le main et `#include <stdio.h>`,`#include <stdlib.h>` en c etc...)

Les langages acceptés seront : `c`, `java`, `python`, `tex`.

Exercice 3 — Make à la main

Faites une commande `makeFST` mimant le make : elle lira un fichier `makefile` donnée en argument (ou nommé `makefile` par défault), le lira en repérant les règles qui sont nécessairement du type :

```
cible : fichierRequis1 fichierRequis2 ...
  Commande1 option cible fichierRequis1 fichierRequis2 ...
```

Pour chaque règle, la commande `makeFST` cherchera si les fichiers `cible` et `fichierRequis` existent.

- Si la `cible` n'existe pas, `makeFST` exécute la `commande1`.
- Si l'un des `fichiersRequis` est plus jeune que la `cible`, elle cherchera si `fichierRequis` est la cible d'une règle. Si oui, elle exécutera cette règle. Sinon elle exécutera la `commande1`.
- Si aucun des `fichierRequis` est plus récent que le fichier cible, alors on exécutera quand même la `commande1`.