Travaux Dirigés de C avancé n°3

Encore un peu d'arguments variables, de la structuration ...

1 Exercice 1 (print it again)

Ecrire et tester une fonction *multiprint* permettant d'afficher des chaînes de caractères (plusieurs fois éventuellement) dans un ordre spécifié par l'utilisateur à l'aide d'une chaîne qui spécifie le format du texte à afficher (à la manière de *printf*). Plus précisément, le prototype de la fonction sera :

```
int multiprint(char * format, ...);
```

où format contiendra :

- des caractères convientionnels, que l'on affichera conformément à leur position ;
- des nombres précédés de % indiquant qu'il faut afficher la n^{ieme} chaîne de caractères passée en argument (%1 désignant la première chaîne après format) ;
- on utilisera la séquence %% pour afficher un %.

Voici un exemple pour conclure :

```
multiprint(``%1 et %1 = %2\n%%1 = %1\n'', ``deux'', ``quatre'');
```

affiche :

```
deux et deux = quatre
%1 = deux
```

Enfin, la fonction *multiprint* devra retourner le nombre de caractères écrits.

2 Exercice 2 (interface graphique)

En utilisant, par exemple, la bibliothèque graphique **MlvLib**, écrire une interface graphique pour le petit jeu développé lors des précédentes séances de TP. Par défaut, le jeu utilisera l'interface en mode texte, l'option `-x` permettant à l'utilisateur de demander l'interface graphique. On notera que tout ce qui est demandé est d'afficher le plateau de jeu et de pouvoir récupérer la case jouée par un joueur humain au moyen d'un clic de souris. On pourra également afficher le message de la victoire à l'aide de l'interface graphique.

Vous trouverez à l'adresse `http://igm.univ-mlv.fr/~oblanc/mlvlib-demo.c`, un programme de démonstration documenté pour la librairie MlvLib à l'aide duquel vous pourrez vous familiariser avec l'utilisation de cette librairie.

3 Exercice 3 (restructuration et compilation)

On souhaite maintenant s'assurer d'une bonne structuration du code. Votre petit jeu devrait se composer des fichiers suivants :

- **plateau.c** et **plateau.h** qui contiennent les types et les fonctions pour la gestion du plateau de jeu indépendamment de l'interface ;
- **strategies.c** et **stratégies.h** qui contiennent les types et les fonctions pour faire jouer l'ordinateur ;
- **text_interface.c** et **text_interface.h** contiennent les fonctions relatives à l'interface en mode texte ;
- **x_interface.c** et **x_interface.h** pour les fonctions relatives à l'interface graphique ;
- **jeu.c** qui contient la fonction *main*, la gestion des options et la boucle principale du jeu.

Une fois assurés de la bonne structuration de votre code, vous réaliserez une **Makefile** avec les contraintes suivantes :

- les cibles *all*, construisant le programme et *clean*, supprimant tous les fichiers objets et temporaires devront exister. La cible *all* sera la cible par défaut.
- Une bibliothèque statique **libmorpion.a** devra être générée à partir de **plateau.c** et **stratégies.c** (on utilisera pour ce faire la commande **ar**). Cette bibliothèque sera utilisée pour la compilation du jeu.

4 Exercice 4 (restructuration v2)

On se retrouve rapidement avec un nombre impressionant de fichiers, nous allons donc les organiser en les répartissant dans différents sous-répertoires afin d'améliorer la lisibilité de l'ensemble. Les règles à respecter sont les suivantes :

- à la racine de votre arborescence ne doit se trouver que votre **Makefile** ;
- tous vos fichiers sources (fichiers **.c**) devront se trouver dans un sous-répertoire **src/** ;
- tous vos fichiers d'en-tête (**.h**) devront se trouver dans un sous-repertoire **include/** ;
- la bibliothèque **libmorpion.a** devra se trouver dans un répertoire **lib/** ;
- et le programme de votre jeu dans le sous-répertoire **bin/**.

Réécrivez votre makefile de façcon appropriée.