

Examen de *Environnements et Outils de Développement*

Note : Vous avez 2 heures.

Vous avez droit à tous documents. La première partie de l'examen est à rédiger sur papier, la deuxième à l'ordinateur. Pour rendre la deuxième partie vous devez, avant la fin de l'examen, **envoyer un mail à zack@pps.univ-paris-diderot.fr** ayant en pièce jointe une **archive tar** du contenu requis. Vous pouvez consulter le Web pour la deuxième partie, mais toutes formes de communication avec les autres sont interdites.

Première partie

Exercice 1 (Le processus de compilation) Répondez aux questions suivantes, *en détail*, et justifiez vos réponses.

1. Décrire le processus de compilation typique du langage C. Détailler les différentes phases de "compilation", en expliquant la relation entre preprocessing, compilation, et liaison.
2. Expliquer la notion de symbol pour le langage de programmation C. Quelle est la différence entre liaison statique et liaison dynamique ? Quels sont les avantages et les inconvénients de deux types de liaison ?

Exercice 2 (Preprocessor) Le fichier `hello.h` contient le code suivant :

```
#define PI 3

#ifdef DEBUG
#define MSG(s) printf("debug: %s\n", s);
#else
#define MSG(s)
#endif
```

```
void hello(void);
void bye() { printf("Bye!\n"); }
```

1. le fichier `hello.c` :

```
#include "hello.h"

void hello() {
    MSG("enter hello")
    printf("Hello, %d world!\n", PI);
    MSG("exit hello")
}
```

montrez le code C disponible à la sortie du preprocessor juste avant la compilation. (Vous pouvez ignorer les annotations de numéro de ligne.)

2. Vous voulez maintenant habilitier l'affichage des messages de debugging. Comment pouvez vous le faire, à l'aide du preprocessor ?
3. Considérez maintenant l'hypothèse d'ajouter à votre projet le fichier `main.c` suivant :

```
#include "hello.h"
#include "hello.c"

int main(void) {
    MSG("enter main")
    hello();
    MSG("exit main")
}
```

que se passera-t-il lors de la compilation du `main.c` ? Expliquer le résultat. Comment pouvez vous résoudre le problème en touchant seulement `hello.h` ?

Justifiez vos réponses.

Exercice 3 (Make) 1. Quel est l'utilité de Make et de Makefile ? Dans le cadre de Make, expliquer les notions de : règle, cible, prérequis, et commande.

2. Considérez le `Makefile` suivant :

```
foo: main.o foo.o bar.o
    gcc -o foo $^
main.o: main.c foo.h
    gcc -c main.c
foo.o: foo.c foo.h bar.h
    gcc -c foo.c
bar.o: bar.c bar.h
    gcc -c bar.c
clean:
    rm -f foo main.o foo.o bar.o
```

donnez le diagramme de dépendances correspondant (i.e. un graphe où chaque noeud correspond à un target et a comme fils ses prérequis)

3. Si aucun des fichiers `*.o` est présent sur disque (et si tous les fichiers `*.c` existent), que se passe-t-il lors de l'exécution de `make foo` ? Donnez la liste des commandes que `make` exécutera, dans un ordre valide.
4. Si, après une première compilation complète, vous modifiez `bar.h` et ensuite exécutez à nouveau `make`, que se passera-t-il ? Donnez à nouveau la liste des commandes que `make` exécutera

Exercice 4 (Gestion de versions) 1. d'après vous, quel est l'intérêt des systèmes de gestion de versions ? Quels sont les avantages par rapport à ne pas les utiliser dans la gestion d'un projet ? Y a-t-il des inconvénients ?

2. Expliquer les différences principales entre un système de gestion de version centralisée (comme Subversion) et un système distribué (comme Git).
3. Expliquer la notion de conflit. Étant donné le fichier textuel `list.txt` suivant :

