Université Paris Diderot
License 3 Informatique

Conduite de Projet
Année 2015–2016

Examen

18 mai 2016

Instructions Motivez vos réponses. Documents autorisés : 2 feuilles A4 recto-verso (4 pages). Tout dispositif électronique et/ou de communication est interdit. La durée de l'examen est de 3 heures.

Exercice 1 [Le processus de compilation, 4 points]

a) Décrivez le processus de compilation typique du langage C. Détaillez les différentes phases, en expliquant la relation entre *preprocessing*, compilation, et édition des liens (*linking*).

b) Quelle est la différence entre liaison statique et liaison dynamique ? Quels sont les avantages et les inconvénients de deux types de liaison ?

Exercice 2 [Preprocesseur, 3 points] Le fichier `gcd.h` contient le code suivant :

```
#ifndef __GCD_H__
# define __GCD_H__
# define TESTSUITE 2
# if TESTSUITE == 1
#  define TEST_NO_A 1
#  define TEST_NO_B 2
# else
#  define TEST_NO_A 37
#  define TEST_NO_B 42
# endif
int gcd(int, int);
void my_print(int x){ /* TODO */ }
#endif /* __GCD_H__ */
```

le fichier `main.c` :

```
#include "gcd.h"
int main(void) {
  int m, n;
  m = TEST_NO_A;
  n = TEST_NO_B;
  my_print(gcd(m, n));
  return 0;
}
```

et le fichier `gcd.c` :

```
#include "gcd.h"
int gcd(int m, int n) {
  int t;
  while (m > 0) {
    if (m < n) { t = m; m = n; n = t; }
    m -= n;
  }
  return n;
}
```

a) Montrez le code C obtenu à la sortie du preprocesseur pour les fichiers `main.c` et `gcd.c`.

b) Que se passerait-t-il lors de la compilation et liaison de `main.c` avec `gcd.c` si on supprime les lignes qui commencent avec `#ifndef` et `#endif` de `gcd.h` ?

Justifiez vos réponses.

Exercice 3 [*Make*, 4 points]

a) Quelle est l'utilité de l'outil Make et des Makefiles ? Dans le cadre de Make, expliquez les notions de : règle, cible, prérequis, et commande.

b) 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 $<
clean:
        rm -f foo main.o foo.o bar.o
```

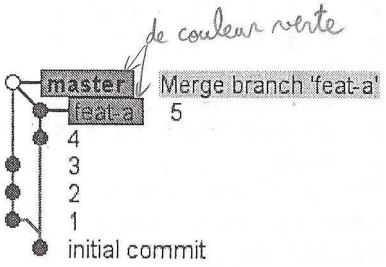
montrez le diagramme de dépendances correspondant (i.e., un graphe dans lesquel chaque noeud corresponde à une cible et les arêtes lient chaque cible à ses prérequis).

c) Si aucun fichier `*.o` est présent (et si tous les fichiers `*.c` existent), que se passera-t-il lors de l'exécution de `make foo` ? Donnez la liste des commandes exécutées par `make`, dans un ordre valide.

d) Si, après une exécution complète de `make foo`, vous modifiez `bar.h` et ensuite exécutez `make`, que se passera-t-il ? Donnez à nouveau la liste des commandes exécutées par `make`, dans un ordre valide.

Exercice 4 [Git, 4 points]

Écrivez une liste des commandes `git` qui produisent un dépôt Git dont l'historique ressemble le plus possible à celui montré en figure—chaque noeud correspond à un commit, les textes sur la droite à des messages de commit, les étiquettes vertes à des noms des branches. Le contenu du dépôt en terme des fichiers n'est pas important (p.ex. vous pouvez utiliser un dépôt qui contient un seul fichier `toto.txt`), mais les *commit*, branches, et *merge* doivent ressembler à ceux en figure.



Exercice 5 [Tests, 5 points]

a) Expliquez l'approche *Test Driven Development* (développement piloté par les tests) au développement de logiciel. Quels sont les avantages et les inconvénients qu'il apporte ?

b) Écrivez l'interface C minimale, dans le format d'un fichier `.h`, d'une structure de données table de hachage qui associe chaînes de caractères à chaînes de caractères. Proposez au moins 10 tests unitaires pour cette structure de données, en donnant une courte description (= 1 phrase) de chaque test. P.ex., « si on insère l'association clef-valeur ⟨`"foo"`, `"bar"`⟩ dans la table, la recherche par clef de `"foo"` doit retourner `"bar"` ».

c) Montrez l'implémentation avec la librairie de test Check d'au moins 5 parmi les tests unitaires proposés.