Examen #2
INF3135

Date: Mercredi, 21 décembre 2005.

Nom: _____
Code permanent: _____

/	3
/	5
/	4
/	4
/	10
/	10
/	4
/	40

Numéro 1. (3 pts) Nous voulons construire un logiciel qui soit robuste. Pourquoi devrait-on utiliser des assertions durant la construction de ce logiciel, et ce même si les assertions ne sont pas activées lors de la compilation finale (en définissant la constante NDEBUG)?

Numéro 2. (5 pts) Un programmeur a conçu un type définissant une classe d'objets pour des `Pile`s de la façon suivante :

```
typedef struct _pile Pile, *PPile;
typedef void * Element;

PPile nouvellePile();
int vide( Pile *p );
void ajouterElement( void *e, PPile pile );
void depilerPile( PPile p );
Element sommet( struct _pile *pile );
void remplacerSommet( PPile p , Element e );
```

Cette description de l'interface du module et des en-têtes des routines pourrait causer des problèmes aux futurs utilisateurs. Réécrivez l'interface de ce module en corrigeant les problèmes tels que le choix des identificateurs, le manque d'uniformité, l'utilisation d'un type non opaque, etc.

Numéro 3. (4 pts) Nous voulons construire un logiciel qui va analyser des programmes en langage 'C'. Plus spécifiquement, ce logiciel va trouver les identificateurs contenus dans un logiciel et les classer dans différents `Dictionnaire`s : un `Dictionnaire` pour les noms de variable, un autre pour les noms de routine, un autre pour les constantes, etc. Ce logiciel va aussi contenir un module, que nous appellerons `Parcoureur`, qui va parcourir le code en entrée et qui va déterminer si, à un point donné du parcours du programme en entrée, la position à ce point correspond à une chaîne de caractères, à un commentaire, ou à divers autres états possibles intéressants. Cet état du parcours sera conservé dans une structure interne au module `Parcoureur`, mais pourra être observé à l'aide d'une opération approprié (`etatPositionCourante()`).

On sait qu'il existe trois (3) principaux types de modules : bibliothèque de routines, machine abstraite et type abstrait.

a) (2 pts) Lequel de ces types de module serait le plus adapté pour le module `Dictionnaire`? Expliquez brièvement pourquoi.

b) (2 pts) Lequel de ces types de module serait minimalement nécessaire pour le module `Parcoureur`? Expliquez brièvement pourquoi.

Numéro 4. (4 pts) Lorsqu'une erreur survient durant l'exécution d'un logiciel, il est possible soit de poursuivre l'exécution, soit de la terminer. Donnez les avantages, inconvénients et les contextes d'utilisations des deux (2) approches.

Numéro 5. (10 pts) On veut définir un module pour simuler un écran graphique. Ce module utilisera les structures de données suivantes :

```
typedef struct _ecran Ecran;
typedef struct _pixelCouleur PixelCouleur;

struct _ecran {
   int nbLignes;
   int nbColonnes;
   PixelCouleur **pixels;
};

struct _pixelCouleur {
   unsigned char rouge;
   unsigned char vert;
   unsigned char bleu;
};
```

La routine suivante permet d'afficher un point à l'écran :

```
void afficherPixel( Ecran ec, int x, int y, unsigned char r, unsigned char v, unsigned char b ) {
   PixelCouleur p;

   // Assertions à compléter...

   p.rouge = r;
   p.vert = v;
   p.bleu = b;

   ec.pixels[x][y] = p;
}
```

a) (4 pts) Complétez cette routine en ajoutant les assertions nécessaires pour vérifier les arguments en entrée.

b) (3 pts) Voici une deuxième routine qui affiche un rectangle à l'écran :

```
void afficherRectangle( Ecran ec, int x1, int x2, int y1, int y2, PixelCouleur c ) {
   int i;

   for( i = x1; i <= x2; i++ ) {
      afficherPixel( ec, i, y1, c.rouge, c.vert, c.bleu );
   }
   for( i = x1; i <= x2; i++ ) {
      afficherPixel( ec, i, y2, c.rouge, c.vert, c.bleu );
   }
   for( i = y1; i <= y2; i++ ) {
      afficherPixel( ec, x1, i, c.rouge, c.vert, c.bleu );
   }
   for( i = y1; i <= y2; i++ ) {
      afficherPixel( ec, x2, i, c.rouge, c.vert, c.bleu );
   }
}
```

Réécrivez une version optimisée de cette routine.

c) (3 pts) Voici une routine utilisant `afficherPixel` et qui permet de construire une division graphique de l'écran :

```
void diviserEnQuatre( Ecran ec ) {
   int i;

   for( i = 0; i < ec.nbColonnes; i ++ ) {
      afficherPixel( ec, i, ec.nbLignes / 2, 0, 0, 0 );
   }
   for( i = 0; i < ec.nbLignes; i ++ ) {
      afficherPixel( ec, ec.nbColonnes / 2, i, 0, 0, 0 );
   }
}
```

Comme le démontre cet exemple, en termes de couplage, la routine `afficherPixel` n'est pas très élégante. Comment pourrait-on modifier l'interface ainsi que le code de la routine afficherPixel de façon à réduire le couplage entre cette routine et les routines appelantes? Quel serait alors le code résultant pour la procédure `diviserEnQuatre`? Réécrivez le code pour expliquer votre réponse. (Vous pouvez évidemment introduire des routines addtionnelles.)

Numéro 6. (10 pts) Voici le code d'une routine qui permet de trouver un élément dans une sous-section d'un tableau de caractères :

```
int trouverElement( char *tab, int debut, int fin, char element ) {
   int res;
   int estTrouve = 0;
   int i = debut;

   while( !  estTrouve && i <= fin) {
      if( tab[i] == element ) {
         estTrouve = 1;
         res = i;
      }
      i++;
   }

   if( !  estTrouve ) {
      res = -1;
   }

   return res;
}
```

Nous voulons écrire le code pour une routine de test `testerTrouverElement`.

a) (4 pts) Tracez le graphe représentant les chemins possibles de la routine `trouverElement`.

b) (3 pts) Décrivez les différents tests découlant de l'application de la méthode des chemins de base. (description : entrées, sorties ...)

c) (3 pts) Écrivez la routine `testerTrouverElement` — qui ne reçoit aucun argument et qui ne produit aucun résultat ou aucune sortie, sauf si une erreur est effectivement détectée (donc qui utilise des assertions pour les divers tests) — qui va effectuer les tests décrits au numéro précédant.

Numéro 7. (4 pts) Soit le script *mystere* dont le code est le suivant :

```
#!/bin/csh

if ($#argv != 1) then
            echo "Usage :  $0 arg1"
            exit 1
endif
foreach fic (`ls $1`)
            if (-d $1/$fic) then
                        ./$0 $1/$fic
            else
                        wc -l $1/$fic
            endif
end
```

a) (2 pts) Indiquez en quelques mots, et sans paraphraser le code, ce que fait ce script.

b) (2 pts) Donnez la ligne de commande permettant d'exécuter le script puis de trier les lignes du résultat par ordre croissant.