Examen final
INF-3135

Date: Jeudi, 27 juillet 2006.

Nom: _____
Code permanent: _____

/	20
/	24
/	10
/	27
/	3
/	16
/	100

Directives pédagogiques :

- Tous les documents sont autorisés.
- Dans toutes les questions de programmation, vous pouvez faire appel sans réserve aux fonctions de la bibliothèque standard, qui incluent entre autres :
  - `stdio.h : sscanf, sprintf, scanf, sprintf, ...`
  - `string.h : strcat, strcpy, strcmp, strlen, ...`
  - `stdlib.h : malloc, free, ...`

Numéro 1. (20 pts) Nous voulons construire une structure de graphe. Pour se faire nous allons utiliser une structure de liste pour les arcs de chaque noeud et une structure d'ensemble pour l'ensemble des noeuds du graphes. Nous voulons que notre module de graphe soit une classe d'objets.

a) (5 pts) Quelle sorte de module serait-il préférable d'utiliser pour la structure de liste. Justifiez votre réponse.

1. Collection de valeurs.
2. bibliothèque de routines.
3. Machine abstraite.
4. Classe d'objets.

b) (10 pts) Voici une proposition pour l'interface (header .h) du module de graphe :

```
#ifndef GRAPHE_H
#define GRAPHE_H

#include "Listes.h"
#include "Ensembles.h"
#include "Exceptions.h"

typedef struct _graphe *Graphe;
typedef void *Element;

void creerGraphe( Graphe *nouveauGraphe );
void detruire_Graphe( Graphe *graphe );

int GrapheVide( Graphe graphe );
int NombreNoeudDansGraphe( Graphe graphe );

void insererNoeud( Element, Graphe *graphe, Exception noeudDejaExistant );
void InsererArc( Exception noeudInexistant, Graphe graphe, Element depart, Element fin );
void supprimerNoeud( Graphe g, Element e, Exception NOEUD_INEXISTANT );
void supprimer_Arc( Graphe graphe,
                    Element noeud_fin, Element noeud_depart,
                    Exception arcInexistant );

#endif
```

Cette interface n'est pas très présentable. Sans modifier les fonctionnalité du module, réécrivez cette interface afin de corriger les signatures des routines (identificateur, ordre des arguments, ...).

c) (5 pts) En plus des modules de listes de d'ensemble cette interface utilise un module d'exception. Ce module va gérer le passage des messages d'erreurs (cas exceptionnels) de l'appelé à l'appelant. Quelle sorte de module serait-il minimalement préférable d'utiliser pour la structure d'exception. Justifiez votre réponse.

1. Collection de valeurs.
2. bibliothèque de routines.
3. Machine abstraite.
4. Classe d'objets.

Numéro 2. (24 pts) Voici un interface proposé pour une collection de valeurs permettant la manipulation de chaîne de caractères.

```
#ifndef CHAINES_H
#define CHAINES_H

typedef struct _chaine *Chaine;

Chaine creerChaineVide();
Chaine creerChaine( char * );
Chaine creerCopie( Chaine );
void detruireChaine( Chaine );

void imprimerChaine( Chaine );
int sontEgal( Chaine, Chaine );
int sontEgalChar( Chaine, char * );
int sontEgalSansTenirCompteDesMajuscules( Chaine, Chaine );

Chaine versMajuscule( Chaine );
Chaine versMinuscule( Chaine );
Chaine composer( Chaine depart, ChaineFin );

#endif
```

Indiquez le type de cohésion que nous avons pour les routines suivantes (justifiez votre choix) :

a) (6 pts) creerChaine

b) (6 pts) sontEgal

c) (6 pts) sontEgalChar

d) (6 pts) sontEgalSansTenirCompteDesMajuscules

Numéro 3. (10 pts) Voici une procédure qui calcule la racine carré d'un nombre (x) plus grand que 1. Le résultat obtenu sera comprit dans l'intervalle suivant :

x - precision $\leq$ resultat * resultat $\leq$ x + precision

Ajouter les assertions nécessaires pour vérifier les pré-conditions et post-conditions de cette routine. (L'argument 'précision' doit toujours être une valeur positive.)

```
double racineCarre( double x, double precision ) {
  double superieur = x;
  double inferieur = 1.0;
  double resultat = 1.0;
  int trouve = 0;

  // insérez vos pré-conditions ici :









  while( ! trouve ) {
    double carre = 0.0;

    resultat = ( superieur + inferieur ) / 2.0;
    carre = resultat * resultat;

    if( carre < ( x - precision ) ) {
      inferieur = resultat;
    } else if( carre > ( x + precision ) ) {
      superieur = resultat;
    } else {
      trouve = 1;
    }
  }

  // insérez vos post-conditions ici :









  return resultat;
}
```

Numéro 4. (27 pts) Voici le code d'une routine d'initialisation de tableau à deux (2) dimensions :

```
void initialiseMatrice( int **matrice, int largeur, int hauteur, int valeurInitiale ) {
  int x = 0;
  int y = 0;

  assert( matrice != NULL );
  assert( hauteur > 0 );
  assert( largeur > 0 );

  for( y = 0; y < hauteur; y++ ) {
    for( x = 0; x < largeur; x++ ) {
      matrice[x][y] = valeurInitiale;
    }
  }
}
```

Nous voulons construire les tests de couverture des chemins "boîte blanche" pour cette routine.

a) (10 pts) Tracez le graphe de couverture des chemins pour cette routine.

b) (5 pts) Calculez la compexité cyclomatique associée à cette routine.

c) (8 pts) Donnez une famille minimale de chemins permettant de tester tous les parcours possibles du graphe.

d) (4 pts) Donnez les cas de tests correspondant à chaque chemin possible.

Numéro 5. (3 pts) Question pour voir qui écoute en classe. Pour quel type de maintenance une compagnie n'investit-elle pas d'argent?

Numéro 6. (16 pts) Soit le script 'csh' suivant :

```
#

if( $#argv == 0 ) then
  foreach nom ( `ls *.h` )
    echo "--- $nom"
    grep $nom *.h
    grep $nom *.c
  end
else
  foreach nom ( $argv )
    echo "--- $nom"
    grep $nom *.h
    grep $nom *.c
  end
endif
```

a) (8 pts) Décrivez ce que fait ce script.

b) (8 pts) Donnez au moins un exemple d'utilisation (c.-à-d. un contexte où le script serait utile).