Master Informatique
Spécialité STL
Parcours LS

Master STIC, mention informatique
Spécialité MOCS
Parcours Logiciels Sûrs

Examen du 27 mars 2006

Tous documents autorisés. Durée 2h.

1 Tests Structurels

Exercice 1

Ce programme permet de calculer l'accéleration maximale prévisible d'un véhicule.

Spécification détaillée

Tout d'abord voici une spécification détaillée sous forme de pseudo-code :

```
Sélection des limites de jerk selon l'état opérationnel
SI PI_reset demandé ALORS
   mettre l'accélération mesurée dans l'accélération de référence
SINON
   SI l'accélération de référence est différente de l'accélération totale ALORS
      SI l'accélération de référence est inférieure à l'accélération totale ALORS
         incrémenter l'accélération de référence par jerkLimit
         SI l'accélération de référence est supérieure à l'accélération totale ALORS
            mettre l'accélération totale dans l'accélération de référence
         FINSI
      SINON
         décrémenter l'accélération de référence par jerkLimit
         SI l'accélération de référence est inférieure à l'accélération totale ALORS
            mettre l'accélération totale dans l'accélération de référence
         FINSI
      FINSI
   FINSI
FINSI
```

NOTE : Le *Jerk* est une unité d'accélération. Il représente la variation d'accélération maximum acceptable pour une période fixée (souvent un cycle d'application du système à surveiller).

Programme

```
enum T_state ( STATE_1, STATE_2, STATE_3, STATE_4) ;
#define JERKLIMITAPPUP  4
#define JERKLIMITAPPDN  2
#define JERKLIMITMOTUP  5
#define JERKLIMITMOTDN  3

extern enum T_State operState;
```

```
extern bool pIReset;
extern bool parkBrake;

void jerkLimiting(int accelerationMesuree, int *accelerationRef)
{

  int jerkLimitUp, jerkLimitDn;

  if((STATE_1 == operState ) ||                                  /* 1 */
     (STATE_2 == operState ))
  {
      jerkLimitUp = JERKLIMITAPPUP;
      jerkLimitDn = JERKLIMITAPPDN;
  }
  else
  {
      jerkLimitUp = JERKLIMITMOTUP;
      jerkLimitDn = JERKLIMITMOTDN;
  }

    if((TRUE == pIReset) &&                                       /* 2 */
     (FALSE == parkBrake))
  {
      *accelerationRef = accelerationMesuree;
  }
  else
  {
    if( *accelerationRef != ACCELERATIONTOTAL )                  /* 3 */
    {
      if( *accelerationRef < ACCELERATIONTOTAL )                 /* 4 */
      {
        *accelerationRef = *accelerationRef + jerkLimitUp;
        if( accelerationRef >  ACCELERATIONTOTAL)                /* 5 */
          *accelerationRef =  ACCELERATIONTOTAL ;
      }
      else
      {
        *accelerationRef = *accelerationRef - jerkLimitDn;
        if( *accelerationRef <  ACCELERATIONTOTAL)               /* 6 */
          *accelerationRef =  ACCELERATIONTOTAL;
      }
    }
  }
}
```

Question 1

Déterminer les entrées et sorties du programme.

Question 2

Dessiner le graphe de contrôle de programme (organigramme).

Question 3
Déterminer les jeux de tests pour obtenir la *couverture des instructions* du programme.

Question 4
éterminer les jeux de tests pour obtenir la *couverture des chemins* du programme.

Question 5
Déterminer les jeux de tests pour obtenir la *couverture des conditions* (circulation) du programme.
Notes :
- *Les commentaires associés à chaque test doivent être explicites et faire référence aux parties testées.*
- *Pour justifier la couverture de test, tous les jeux de tests doivent être décrits, y compris les jeux de tests impossibles.*

Exercice 2
La fonction `CompareData` permet de déterminer si 2 données sont égales à une tolérance près.

Programme
```
typedef enum Bool (FALSE = 0, TRUE);

static Bool CompareData( int data1, int data2, int tolerance)
{
   Bool retVal;

   if (data1 >= data2)
   {
      if (retVal = (data1 <= (data2 + tolerance)))
        retVal = TRUE
      else
        retVal = FALSE;
   }
   else
   {
      if ((data1 + tolerance) >= data2)
        retVal = TRUE
      else
        retVal = FALSE;
   }

   return(retVal);
}
```

Question 1
Déterminer les entrées et sorties du programme.

Question 2
Déterminer les domaines fonctionnels et informatiques des entrées et sorties du programme.

Question 3
Dessiner le graphe de contrôle de programme (organigramme).

Question 4
Déterminer les jeux de tests pour obtenir la *couverture des données aux limites* du programme.

2 Tests Fonctionnels

Exercice 3

Il s'agit de tester la commande Unix `dir` dont voici ci-dessous un extrait du manuel d'utilisation.

```
LS(1L)                  Manuel de l'utilisateur Linux                  LS(1L)


NOM
       dir - Afficher le contenu d'un répertoire.

SYNOPSIS
       dir [fichier...]

       Options : [-aC] [-T cols] [--]

DESCRIPTION
       La commande dir affiche tout d'abord l'ensemble de ses arguments
       fichiers autres que des répertoires. Puis dir affiche l'ensemble des
       fichiers contenus dans chaque répertoire indiqué. Si aucun argument
       autre qu'une option n'est fourni, l'argument '.' (répertoire en cours)
       est pris par défaut. Un fichier n'est affiché que si son nom ne commen-
       ce pas par un point, ou si l'option -a est fournie.

       Chacune des listes de fichiers (fichiers autres que des répertoires, et
       contenu de chaque répertoire) est triée séparément par ordre alphabéti-
       que.

       Le résultat est envoyé sur la sortie standard, un élément par ligne,
       sauf si un affichage multi-colonnes est demandé avec l'option -C.

OPTIONS
       -a
              Afficher tous les fichiers des répertoires, y compris les
              fichiers commençant par un '.'.

       -C
              Présenter les fichiers en colonnes, triés verticalement.

       -T cols
              Supposer que les tabulations sont espacées de cols colonnes. La
              valeur par défaut est 8, mais elles peut être modifiée par la
              variable d'environnement TABSIZE. dir utilise des tabulations
              pour accélérer l'affichage, mais si cols vaut zéro, aucune tabu-
              lation ne sera employée.

       --     Terminer la liste des options.

ENVIRONNEMENT
       La variable d'environnement TABSIZE détermine le nombre de caractères
       par saut de tabulation. La variable COLUMNS (lorsqu'elle contient un
       entier décimal) déterminer le nombre de colonnes pour la largeur de
       sortie (option -C).

TRADUCTION
       Christophe Blaess, 1997.
```

FSF 22 octobre 2002 DIR(1L)

Question 1
Déterminer les entrées et sorties de la commande.

Question 2
Déterminer des jeux de tests pour obtenir la *couverture nominale* de la commande.

Question 3
Déterminer des jeux de tests pour obtenir la *couverture aux* et *hors limites* la commande.