Examen

12 janvier 2023 — Durée 2h

Seul document autorisé : Mémento C sans annotation

Les trois parties sont indépendantes et peuvent être traitées dans un ordre quelconque.

On rappelle en annexe la spécification des paquetages `terrain`, `robot` et `environnement` utilisées dans les trois parties de ce sujet, ainsi que le format des terrains lus par la fonction `lire_terrain`. L'implémentation de ces paquetages n'est pas utile pour ce sujet : il est demandé, notamment pour les tests, de raisonner en «boîte noire».

Partie I : Tests fonctionnels (11 pt)

Exercice 1. (1 pt) Qu'est-ce qu'un test *fonctionnel* ?

Exercice 2. (1 pt) Donner un exemple de terrain simple, dans un format lisible par la fonction `lire_terrain` du paquetage `terrain`.

Exercice 3. (4 pt) On souhaite écrire un programme `testenvironnement.c` permettant de tester les fonctions `lire_terrain` du paquetage `terrain`, et `mesure_envt` du paquetage `environnement`.

Écrire un programme de test lisant un *fichier de test* dont le nom est donné en argument de la ligne de commande. Le format de ce fichier de test sera :

- sur la première ligne, le nom d'un fichier terrain ;
- sur la deuxième ligne, le paramètre correspondant à la direction de la mesure (de 0 à 8, cf la spécification de la fonction `mesure_envt`)
- sur la troisième ligne, le résultat attendu (0, 1 ou 2) de la fonction `mesure_envt`, depuis la position initiale du robot, avec le paramètre indiqué.

Ce programme doit afficher un message indiquant si le test est réussi ou non.

Exercice 4. (2 pt) Écrire un fichier `Makefile` permettant de compiler le programme écrit pour l'exercice précédent. L'exécution de la commande `make` sans argument doit générer un exécutable nommé `testenvironnement`.

Exercice 5. (3 pt)

1. Décrire un jeu de tests fonctionnels permettant de tester la fonction `mesure_envt` à l'aide du programme de test écrit à l'exercice 3.
2. Donner un exemple de test fonctionnel de ce jeu de tests, utilisant le terrain donné à l'exercice 2, et dans un format lisible par le programme écrit à l'exercice 3.

Partie II : Tests de robustesse (3 pt)

Exercice 6. (1 pt) Qu'est-ce qu'un test de *robustesse* ?

Exercice 7. (2 pt) Donner deux exemples (différents) de tests de robustesse pour la fonction `mesure_envt`.

Partie III : Programmation (6 pt)

Exercice 8. (6 pt) Écrire un programme simulant un robot aléatoire : ce programme prend en argument de la ligne de commande le nom d'un *fichier terrain*, et déplace le robot de manière aléatoire (en tirant au hasard le fait d'aller tout droit, de tourner à gauche ou de tourner à droite), *jusqu'à être sorti du terrain*, et *sans rentrer dans aucun obstacle*.

Une fois le robot sorti du terrain, le programme termine en affichant le nombre de déplacements effectués.

Annexes

A.1 Fichier terrain.h

```
#ifndef _TERRAIN_H_
#define _TERRAIN_H_
#include <stdio.h>

typedef enum { LIBRE, EAU, ROCHER } Case;

#define DIM_MAX 256

// indexation utilisée :
// 1er indice : abscisse = colonne (colonne de gauche : abscisse = 0)
// 2è me indice : ordonnée = ligne (ligne du haut : ordonnée = 0)

typedef struct {
  int largeur, hauteur;
  Case tab[DIM_MAX][DIM_MAX];
} Terrain;

typedef enum {
  OK,
  ERREUR_FICHIER,
  ERREUR_LECTURE_LARGEUR,
  ERREUR_LECTURE_HAUTEUR,
  ERREUR_LARGEUR_INCORRECTE,
  ERREUR_HAUTEUR_INCORRECTE,
  ERREUR_CARACTERE_INCORRECT,
  ERREUR_LIGNE_TROP_LONGUE,
  ERREUR_LIGNE_TROP_COURTE,
  ERREUR_LIGNES_MANQUANTES,
  ERREUR_POSITION_ROBOT_MANQUANTE
} erreur_terrain;

/* Lecture d'un terrain dans un fichier de nom nom_fichier
   Résultats :
   t le terrain lu
   x, y position initiale du robot lue dans le fichier terrain
   Renvoie :
   OK si la lecture s'est déroulée correctement
   ERREUR_FICHIER si le fichier n'a pas pu être ouvert
   ...
 */
erreur_terrain lire_terrain(FILE * f, Terrain * t, int * x, int * y);

/* Largeur d'un terrain */
int largeur(Terrain * t);

/* Hauteur d'un terrain */
int hauteur(Terrain * t);

/* Indique si la case de coordonnées (x,y) est libre
   Renvoie vrai ssi les 3 conditions suivantes sont vraies :
    - 0 <= x < largeur
    - 0 <= y < hauteur
    - t.tab[x][y] = LIBRE
 */
int est_case_libre(Terrain * t, int x, int y);

void afficher_terrain(Terrain * t);

/* Écrire le terrain T dans le fichier f. (x,y) sont les coordonnées du robot */
void ecrire_terrain(FILE * f, Terrain * T, int x, int y);
#endif
```

A.2 Fichier robot.h

```
#ifndef _ROBOT_H_
#define _ROBOT_H_


typedef enum { Nord, Est, Sud, Ouest } Orientation;

typedef struct {
  int x, y;
  Orientation o;
} Robot;

/* initialiser le robot r en position (x,y) et orientation o */
void init_robot(Robot * r, int x, int y, Orientation o);

/* faire avancer le robot d'une case */
void avancer(Robot * r);

/* faire tourner le robot àgauche */
void tourner_a_gauche(Robot * r);

/* faire tourner le robot àdroite */
void tourner_a_droite(Robot * r);

/* recupere la position de la case du robot */
void position(Robot * r, int * x, int * y);

/* recupere la position en abscisse de la case du robot */
int abscisse(Robot * r);

/* recupere la position en ordonnee de la case du robot */
int ordonnee(Robot * r);

/* recupere l'orientation du robot */
Orientation orient(Robot * r);

/* recupere la position de la case devant le robot */
void position_devant(Robot * r, int * x, int * y);

#endif
```

A.3 Fichier environnement.h

```
#ifndef _ENVIRONNEMENT_H_
#define _ENVIRONNEMENT_H_

#include "robot.h"
#include "terrain.h"

/* Environnement : terrain + robot */

typedef struct {
  Robot r;
  Terrain t;
} Environnement;

/* Initialise l'environnement envt :
   - lit le terrain dans le fichier fichier_terrain
   - initialise le robot : coordonnées initiales lues dans le fichier
   terrain, orientation initiale vers l'est
*/
erreur_terrain initialise_environnement(Environnement * envt, char * fichier_terrain);

/* Résultat d'un déplacement de robot */
typedef enum {
  OK_DEPL, /* Déplacement sur case libre */
  PLOUF, /* Déplacement dans l'eau */
  CRASH, /* Déplacement dans un rocher */
  SORTIE, /* Sortie du terrain */
} resultat_deplacement;

/* Avancer le robot sur le terrain : */
resultat_deplacement avancer_envt(Environnement * envt);

/* Tourner le robot àgauche */
void gauche_envt(Environnement * envt);

/* Tourner le robot àdroite */
void droite_envt(Environnement * envt);

/* Effectuer une mesure
   Paramè tre d : la direction de la mesure
     0 sur place
     1 devant
     2 devant droite
     3 droite
     4 derriè re droite
     5 derriè re
     6 derriè re gauche
     7 gauche
     8 devant gauche
   Renvoie le résultat de la mesure :
     0 rien (case libre ou en-dehors du terrain)
     1 eau
     2 rocher
 */
int mesure_envt(Environnement * envt, int d);

/* Afficher le terrain avec la position et l'orientation du robot */
void afficher_envt(Environnement * envt);

#endif
```

A.4 Format des fichiers terrains

Un *terrain* est un rectangle composé de cases carrées (`L` en largeur et `H` en hauteur), chaque case pouvant être libre (caractère `'.'`), occupée par de l'eau (caractère '~'), ou occupée par un rocher (caractère `'#'`). La case initiale du robot est une case libre, marquée par le caractère `'C'`.

Préconditions pour le terrain :

— les dimensions `L` (largeur) et `H` (hauteur) doivent être inférieures à une dimension maximale `DIM_MAX`,

— il existe un chemin (formé de cases libres) entre la case centrale et l'extérieur du terrain (la sortie).

Un fichier *terrain* est un fichier composé :

1. d'un entier `L`, la largeur du terrain
2. d'un entier `H`, la hauteur du terrain
3. de `H` lignes de `L` caractères dans l'ensemble {`'#','.','~','C'`}.