Université de Nice-Sophia Antipolis — POLYTECH
ELEC3 — 2020–2021

Examen de Langage C

Durée : 1h
Aucun document autorisé
Mobiles interdits *Toutes les fonctions que vous écrirez doivent être clairement commentées avec des affirmations significatives (antécédents, conséquents, invariants). Vous prendrez soin de définir les bons paramètres et les bons types des données manipulées. Pensez à définir des fonctions auxiliaires si cela est nécessaire.*

Exercice 1

Soit une matrice d'entiers $n \times n$, écrivez la fonction `somme` qui renvoie la somme des valeurs des éléments qui se trouvent sur les deux diagonales. Attention aux indices !

Par exemple, pour la matrice 4×4 donnée ci-dessous, la fonction renverra l'entier 68.

$$\begin{vmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{vmatrix}$$

```c
/*
 * Antécédent : n>0
 *              mat matrice n×n d'entiers initialisée
 * Rôle : renvoie la somme des entiers des 2 diagonales de la matrice mat
 */
int somme(const int n, const int mat[][n]) {
  assert(n>0);
  int s = 0;
  for (int i=0; i<n; i++)
    // invariant : s = somme des 2 diagonales entre 0 et i-1
    s += mat[i][i] + mat[i][n-i-1];
  return s;
}
```

Exercice 2

Une matrice $m \times n$ dont les dimensions sont lues sur l'entrée standard contient des entiers quelconques, à l'exception de la case $(0, 0)$ qui contient la valeur 0.

On veut déplacer un jeton sur la matrice de haut en bas et gauche à droite au gré de la valeur d'un dé. On lance répétitivement le dé qui renvoie à chaque fois une valeur comprise entre 1 et 6 correspondant au nombre de cases à avancer. Les tirages du dé s'arrêtent dès qu'on dépasse la dernière case.

Écrivez la procédure `somme` qui prend une matrice `mat` et ses dimensions en paramètres et qui, en partant de la case $(0, 0)$, renvoie la somme des valeurs des cases sur lesquelles le jeton passe. La fonction `lancerDé()` est donnée et renvoie de façon aléatoire un entier compris entre 1 et 6.

Par exemple, pour la matrice 4×3 donnée ci-dessous :

$$\begin{vmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & 11 \end{vmatrix}$$

et avec la suite de tirages suivante $2, 4, 3, 1, 4$, la procédure `somme` renverra 27 (*i.e.* $27 = 2 + 6 + 9 + 10$). Notez que le dernier tirage 4 fait sortir de matrice.

```c
#define MAX_DE 6

/*
 * Antécédent : mat matrice m×n d'entiers initialisée
 *              mat[0][0]==0 et m*n> MAX_DE
 *
 * Rôle :  calcule la somme des cases de la matrice mat
 *         par lesquelles passe un jeton dont le déplacement
 *         en nombre de cases est donné par le jet d'un dé.
 *         Le déplacement se fait de haut en bas et de gauche à droite.
 */
int somme(const int m, const int n, const int mat[m][n]) {
  const int nbCases = m*n;
  assert(nbCases>MAX_DE);
  int som=0, caseCourante=0;
  do {
    // avancer le jeton de lancerDe() cases
    caseCourante+=lancerDe();
    if (caseCourante>=nbCases)
      // on sort de la matrice ⇒ on renvoie la somme totale
      return som;
    // else : on reste dans la matrice ⇒ incrémenter som de la valeur
    // de la case courante
    som+=mat[caseCourante/n][caseCourante%n];
    // Invariant : som est égale à la somme des valeurs des cases
    // par lesquelles le jeton est passé depuis la première case
    // jusqu'à la case courante
  } while (1);
}
```

Exercice 3

RLE (*Run Length Encoding*) est une technique de compression de données, anciennement utilisée pour la compression d'images. Le principe est rudimentaire et très simple : une séquence de caractères c identiques est *remplacée* par :

c **marqueur** L

où **marqueur** est un caractère spécial, si possible, peu fréquent dans la suite de caractères à comprimer et L la longueur de la séquence de caractères c codée sur 1 seul caractère. Si L est codée sur un caractère, cela veut dire que ce codage ne comprime qu'une suite d'au plus 9 caractères identiques. Si la suite fait plus de 9 caractères, les 9 premiers seront comprimés, puis la suite de la séquence sera considérée comme une nouvelle séquence à coder. Lorsque le marqueur apparaît dans les données à comprimer, il est remplacer par « **marqueur** 0 ».

Par exemple, si on choisit comme marqueur le caractère #, la suite `bbbbb##aaaaaaaaaaabb#xxx` sera codée `b#5#0#0a#9a#2b#2#0x#3`.

Écrivez la procédure `decomprimer` qui lit toute l'entrée standard (avec `getchar` uniquement) une suite de caractères comprimée selon la méthode *RLE* avec `'#'` comme marqueur et qui écrit sur la sortie standard (avec `putchar` uniquement) sa forme décomprimée. La marque est passée en paramètre de la procédure.

Par exemple, si l'entrée standard contient `b#5#0#0a#9a#2b#2#0x#3`, la procédure `decomprimer` écrira sur la sortie standard `bbbbb##aaaaaaaaaabb#xxx`.

D'autre part, vous penserez à signaler une erreur si l'entrée standard n'est pas conforme à la syntaxe *RLE*.

```c
/*
 * Antécédent : '0' <= c <= '9'
 * Rôle :  renvoie la conversion du chiffre c en entier
 */
int toInt(const char c) {
  return c-'0';
}
/*
 *  Rôle : écrit le message msg sur la sortie d'erreur standard
 *         et arrête le programme avec code = EXIT_FAILURE
 */
void erreur(char msg[]) {
  fprintf(stderr, "%s\n", msg);
  exit(EXIT_FAILURE);
}
/*
 * Rôle : écrit n fois le caractère c sur la sortie standard
 */
void decoder(const char c, const int n) {
    for (int i=0; i<n ; i++)
      putchar(c);
}
/*
 * Rôle : écrit le marqueur m sur la sortie standard
 */
void decoderMarqueur(const char m) {
  putchar(m);
}
/*
 * Rôle : renvoie le prochain caractère lu sur l'E/S. Signale une erreur si
 *        la fin de fichier est atteinte
 */
```



```c
int nextChar() {
  int c = getchar();
  if (c==EOF) erreur("EOF atteinte");
  //
  return c;
}

/*
 * Rôle : décomprime la suite de caractères lue sur l'entrée standard
 *        selon la technique RLE avec la marqueur mark
 */
void decomprimer(const char mark) {
  const int lgMax = 9;
  int c;
  while ((c=getchar())!=EOF) {
    if (c==mark) {
      // la marque est un simple caractère et doit être suivie par 0
      if (toInt(nextChar())!=0) erreur("0 attendu");
      decoderMarqueur(mark);
    }
    else {
      // traiter un caractère à décoder ⇒ vérifier la présence du marqueur
      if (nextChar()!=mark)
        erreur("marqueur attendu");
      // vérifier la validité de la longueur
      int lg = toInt(nextChar());
      if (lg<1||lg>lgMax)
        erreur("1<=longueur<=9");
      // ok
      decoder(c,lg);
    }
  }
}
```