TP : des dessins en mode texte

Objectif(s)

- ✯ Apprendre à se repérer dans un espace à deux dimensions.
- ✯ Renforcement sur les boucles imbriquées

Exercices obligatoires

Afin de travailler votre aisance avec les boucles imbriquées, nous vous proposons un exercice qui utilise une interface vous permettant de faire des dessins simples en mode texte.

Un canvas est un tableau de caractères à deux dimensions dans lequel on peut mettre un caractère dans une des cases repérées par un numéro de colonne et un numéro de ligne (entier).

En C, le `canvas` est une référence (via un `typedef` vers un pointeur vers la structure comportant les informations essentielles. Vous pouvez donc utiliser `canvas` comme un type de donnée.

Les fonctions pour utiliser les `canvas` sont les suivantes :

— `create_canvas(unsigned int dim)` crée et retourne un canvas (carré) de dimension dim. Les numéros de ligne et de colonne sont compris dans l'intervalle $[0, dim[$. Chaque case du canvas est initialisée avec le caractère `' '` (espace).

— `set(canvas c, int lig, int col, char caractere)` met le `caractere` à la case (`lig`, `col`) du canvas `c`.

— `get(canvas c, int lig, int col)` retourne la valeur du `caractere` de la case (`lig`, `col`) du canvas `c`.

— `get_size(canvas)` retourne la taille du côté du canvas.

— `plot(canvas c)` affiche le canvas `c` dans le terminal, entouré de chiffres indicateurs des coordonnées des lignes et colonnes.

Le fichier `canvas.h` contient les entêtes des fonctions permettant de gérer votre surface de dessin et la documentation de la bibliothèque.

Vous devez télécharger les fichiers `canvas.c` et `canvas.h` qui contiennent les fonctions permettant de gérer votre surface de dessin. Appuyez vous ensuite sur le fichier `demo.c` pour utiliser ces fichiers.

Compilez le fichier `canvas.c` avec la commande :

```
gcc -Wall -c canvas.c
```

Cette compilation n'est normalement à faire qu'une seule fois.

Vous pouvez ensuite compiler le fichier `demo.c` avec la commande :

```
gcc -Wall demo.c canvas.o -o demo
```

Exercice 1

Question 1

Ouvrez dans votre éditeur préféré le fichier `demo.c` Compilez et lancez le programme de demonstration.

Question 2

Sur le modèle du programme de démonstration, écrivez une fonction `damier` qui affiche un damier sur le terminal. Un exemple de résultat attendu

```
.01234567.
0 X X X X0
1X X X X 1
2 X X X X2
3X X X X 3
4 X X X X4
5X X X X 5
6 X X X X6
7X X X X 7
.01234567.
```

Question 3

Écrivez aussi une fonction `damier64` qui affiche un damier sur le terminal, mais de 64 cases de coté.

Question 4

Écrivez une fonction `diagonale` qui affiche une diagonale sur le terminal. Un exemple de résultat attendu :

```
.01234567.
0X       0
1 X      1
2  X     2
3   X    3
4    X   4
5     X  5
6      X 6
7       X7
.01234567.
```

Question 5

Écrivez une fonction `deuxieme_diagonale` qui affiche la deuxieme diagonale sur le terminal. Un exemple de résultat attendu :

```
.01234567.
0       X0
1      X 1
2     X  2
3    X   3
4   X    4
5  X     5
6 X      6
7X       7
.01234567.
```

Question 6

Écrivez une fonction `un_carre` qui dessine un carré de largeur `cote` à une position (`line`, `col`). Le prototype sera :

```
void un_carre(canvas c, int cote, int line, int col)
```

Écrivez une fonction `test_carre` qui

— cree un canvas

— appelle la fonction `un_carre`

— affiche le canvas.

Exemple de résultat attendu pour l'appel `un_carre(c, 5, 2, 1)` :

```
.012345678.
0         0
1         1
2 XXXXX   2
3 X   X   3
4 X   X   4
5 X   X   5
6 XXXXX   6
7         7
8         8
.012345678.
```

Question 7

Écrivez une fonction `cible` qui permet d'obtenir le résultat suivant pour un paramètre n (égal à 16 dans l'exemple). :

```
.0123456789012345.
0XXXXXXXXXXXXXXXX0
1X              X1
2X XXXXXXXXXXX X2
3X X          X X3
4X X XXXXXXX  X X4
5X X X      X X X5
6X X X XXXX X X X6
7X X X X  X X X X7
8X X X X  X X X X8
9X X X XXXX X X X9
0X X X      X X X0
1X X XXXXXXX  X X1
2X X          X X2
3X XXXXXXXXXXX X3
4X              X4
5XXXXXXXXXXXXXXXX5
.0123456789012345.
```

Question 8

Écrivez une fonction `un_rectangle` qui dessine un rectangle en connaissant uniquement les coordonnées de deux points diamétralement opposés.

Exemples d'appels de la fonction :

```
rectangle(c, 3, 4, 10, 11);
rectangle(c, 2, 10, 7, 8);
```

Écrivez une fonction `test_rectangle`.

Question 9

Écrivez une fonction `case_haut_gauche` qui prend un canvas en parametre et permet de récupérer dans des variables les coordonnées de la case la plus en haut à gauche non vide. Pour les points $(2, 1)$ et $(1, 2)$ la valeur retournée par la fonction sera le point $(2, 1)$ (il s'agit en fait de la case la plus à gauche parmi les cases les plus en haut).

Question 10

Écrivez une fonction `miroir` qui prend en parametre un canvas et effectue un mirroir horizontal. Exemple de résultat attendu :

```
.01234567.          .01234567.
0        0          0        0
1     *  1          1  *     1
2  ***   2          2   ***  2
3 *      3          3      * 3
4        4          4        4
5        5          5        5
6        6          6        6
7        7          7        7
.01234567.          .01234567.
```

Attention : il y a quelques differences entre le traitement d'une surface de côté pair ou impair.

Renforcements

Exercice 2

Question 1

Écrivez une fonction qui effectue une rotation de 90° d'une surface.

Question 2

Écrivez une fonction qui dessine un rectangle incliné de 45° avec les coordonnées des coins.

Question 3

Écrivez une fonction qui dessine un disque (un rond plein) dans un canvas pris en paramètre, avec les coordonnées du centre et le rayon du disque.

Question 4

Écrivez une fonction qui dessine un cercle. Éssayez en vous inspirant du disque, vous constaterez que cela ne marche pas très bien. Vous pouvez utiliser un algorithme plus approprié, décris à l'adresse :

`https://fr.wikipedia.org/wiki/Algorithme_de_trac%C3%A9_d%27arc_de_cercle_de_Bresenham)`