TP 2 Programmation – DUT 1 – Définition et tests de fonctions

P.Courtieu

Septembre 2017

1 Rappels

Téléchargez la bibliothèque d'entrée-sortie : inout.h et inout.c et mettez les dans un répertoire `tp2`. Pour cela vous pouvez faire les commandes suivantes (une seule fois) dans un terminal (menu pricipal : `terminal/Konsole`) :

```
mkdir tp2 # si tp2 pas deja cree
cd tp2
wget "http://deptinfo.cnam.fr/~courtiep/inout/inout.h"
wget "http://deptinfo.cnam.fr/~courtiep/inout/inout.c"
```

`BOOL` : Continuez à utiliser un pseudo type `BOOL` à la place de **`int`** :

```
#define BOOL int
#define TRUE 1
#define FALSE 0
```

`BOOL` toujours : Attention Ne faites ***jamais*** **`if`** `(xxx == TRUE)` mais **`if`** `(XXX)`. Symétriquement ne faites pas **`if`** `(xxx==FALSE)` mais **`if`** `(!XXX)`.

Tests : Programmez et tester ces fonctions *une par une* (testez une fonction dès que vous pensez qu'elle est finie). Pour tester une fonction `f` on programme un ou plusieurs appels à cette procédure dans une procédure `test_f` de la manière vue en cours : on test le résultat obtenu par rapport au résultat attendu et on affiche un message d'erreur si ils ne sont pas égaux.

Travail demandé : Si vous êtes débutant il est probable que vous restiez sur le morpion toute la séance, *ce n'est pas DU TOUT un problème*. Si vous connaissez déjà bien les tableaux passez à l'exercice 2 ou 3 directement. **Attention** l'exercice 3 est nettement plus difficile (tableaux à deux dimension + « chute » de la pièce sur une colonne...) ne vous surestimez pas en sautant le 2.

2 Fin TD 1

Ceux qui n'ont pas fini le TD 1, faites le chez vous.

3 Jeu du morpion

On désire écrire un jeu de morpion jouable à deux sur un clavier unique, les joueurs tapent leurs coups alternativement.

Une partie typique devrait ressembler à l'écran à ceci :

```
___
___
___

Joueur1? 5

___
_x_
___

Joueur 2? 1

o__
_x_
___

Joueur 1? 7

o__
_x_
x__

Joueur 2? 3

o_o
_x_
x__

...
```

Plus précisément :

1. Avant chaque coup le programme affiche la grille : 3 lignes de 3 caractères : `'_'` pour la case vide, `'x'` pour un case cochée par le joueur 1, et `'o'` pour une case cochée par le joueur 2.
2. Ensuite le programme affiche le nom du joueur qui doit jouer et lit le coup au clavier : un entier entre 1 et 9.
3. Si le coup est invalide pour une raison quelconque le coup n'est pas exécuté et c'est au joueur suivant de jouer (retour à l'étape 1.).
4. Si le coup est valide et que la partie n'est pas finie le coup est exécuté (une case est cochée) et c'est au joueur suivant de jouer (retour à l'étape 1.).
5. Si le coup est autorisé et gagnant le programme s'arrêter (`exit(0);`) avec un message indiquant quel joueur a gagné.

3.1 La grille du morpion en mémoire

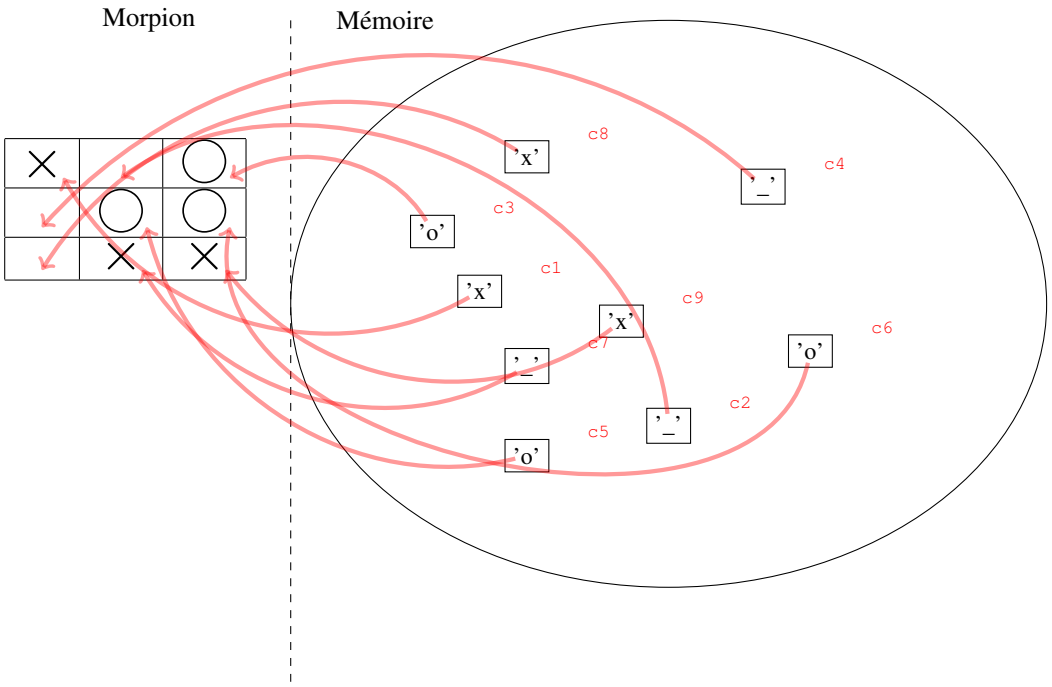
Le squelette de programme fourni est basé sur l'idée suivante : l'état du morpion se résume à l'état des 9 cases de la grille. À tout moment chaque case est dans l'un des 3 états suivants : vide, cochée X ou cochée O.

Pour représenter en mémoire la grille et les trois états possibles de chaque case on utilise 9 variables `c1`...`c9` de type **`char`** déclarées dans la fonction `main`. Ces variables représentent les cases de la grille du morpion : `c1` représente la case 1, `c2` la case 2 etc.

Votre programme doit faire en sorte qu'à tout moment le caractère contenu dans chaque variable correspond à l'état de la case correspondante (`'_'` si la case est vide, `'x'` si elle a été cochée par le joueur 1 et `'o'` si par le joueur 2).

De cette façon pour afficher la grille il suffit d'afficher le contenu des variables, pour savoir l'état de la case i il suffit de regarder le contenu de la variable `ci`.

Le schéma ci-dessous montre d'un côté la grille du morpion telle que l'utilisateur l'imagine et de l'autre telle qu'elle existe en mémoire. On voit que les variables sont dans des adresses aléatoires de la mémoire (différentes à chaque exécution du programme). C'est uniquement l'utilisation (l'affichage, le calcul de la victoire) de ces variables qui permet de donner l'« illusion » que les cases sont organisées en grille. Nous verrons cela en cours bientôt.



3.2 Dans votre fonction principale

Le squelette fourni contient plusieurs fonctions et procédures que vous devez compléter puis utiliser dans le `main`. N'hésitez pas à ajouter des fonctions supplémentaires si vous en avez le besoin.

Vous noterez l'utilisation d'un nouveau type de variable : **`char`**. Il s'agit des variables contenant un caractère (un seul). Pour représenter une valeur constante de ce type on écrit un caractère entre *guillemets simples*, par exemple le caractère « c » s'écrit `'c'`. Autre exemple : dans le squelette vous trouverez le caratère « souligné » `'_'`.

On peut tester l'égalité entre deux caractères avec le test `==`.

```
#include"inout.h"
#define BOOL int
#define TRUE 1
#define FALSE 0

/* Retourne le caractère de la case n dans la grille a1...a9. Si n
   n'est pas compris entre 1 et 9 le comportement de cette fonction
   est indéfini. ATTENTION: cette fonction ne permet pas de modifier
   une case. */
BOOL getCase(int n, char a1,char a2,char a3,char a4,char a5,
             char a6,char a7,char a8,char a9){
// À coder
  return a1;  // INTENTIONNELLEMENT FAUX, à vous de l'écrire
}
```

```
/* Teste si la grille décrite par les 9 caractères est uneposition gagnante. */
BOOL testpasGagne(char a1,char a2,char a3,char a4,char a5,
                  char a6,char a7,char a8,char a9){
  return TRUE; // INTENTIONNELLEMENT FAUX, à vous de l'écrire
}


/* Affiche les neufs caractère comme une grille dans le terminal: 3 lignes de 3 caractères */
void afficheGrille(char a1,char a2,char a3,char a4,char a5,char a6,char a7,char a8,char a9){
// À coder
}

int main() {
  char c1='_', c2='_', c3='_', c4='_', c5='_', c6='_', c7='_', c8='_', c9='_';
  // contiendra le numéro désigné par le joueur qui joue
  int casecochee;
  // doit devenir faux lorsque une victoire est détectée, afin
  // d'arrêter la boucle while ci-dessous
  BOOL pasgagne=TRUE;
  // Le caractère correspondant au joueur dont c'est le tour
  char coche = 'x';
  // Boucle principale: on en sort lorsqu'un jouer gagne. Amélioration
  // possible: on sort aussi lorsque toutes les cases sont cochées.
  while(pasgagne) {
    // À coder:
    // 1) afficher la grille
    afficheGrille(c1,c2,c3,c4,c5,c6,c7,c8,c9);
    // 2) lire un entier au clavier
    ecrireString("Quelle case? ("); ecrireChar(coche); ecrireString(")");
    casecochee = lireInt();
    // 3) Redemander la case jusqu'à obtenir une case existantes et vide.
    // (getCase peut être utile ici)
    // 4) si coup gagnant metre à jour pasgagne
    // (testPasGagne peut être utile ici)
    // 5) changer de joueur (coche devient 'x' (si 'o') et 'o' (si 'x'))
  }
  // Afficher une dernière fois la grille et sortir
/* REMPLIR ICI */
}
```

Le déroulement du jeu est à programmer dans la boucle **`while`** (voir ci-dessous pour une explication). Les différentes étapes sont écrites en commentaire.

3.3 Exercice 1

Programmez le morpion.

3.4 Amélioration

Définissez une fonction `lireCoupMorpion` de lecture d'un entier au clavier qui redemande un coup tant que celui-ci est invalide. Quels arguments doit prendre cette fonction ? Que doit-elle retourner ?

Pour redemander la lecture vous pouvez soit utiliser une boucle **`while`** (voir ci-dessous) soit simplement rappeler la fonction `lireCoupMorpion`.

3.5 Exercice 2

Reprogrammez le jeu de morpion avec un tableau de 9 cases à la place des 9 variables. Pour déclarer ce tableau il suffit d'écrire ceci dans le main :

```
int main() {
  char grille [] = {'_','_','_','_','_','_','_','_','_'};
```

Maintenant la case 1 est représentée par la case 0 du tableau : `grille[0]`. Plus généralement la case `x` est représentée par `grille[x-1]`.

Petite précision : vous pouvez écrire des fonctions qui prennent le tableau en paramètre (à la place de `c1...c9`). *GROS AVANTAGE : vos fonctions peuvent modifier le contenu des cases du tableau passé en paramètre* (en réalité c'est l'adresse du tableau qui est envoyé à la fonction). On peut donc imaginer une nouvelle fonction :

```
void setCase(int i, char val, int[] g) {
  grille[i] = val;
}
```

telle que `setCase(casecochee-1,'x',grille)` met le caractère `'x'` dans la case numéro `casecochee-1` du tableau `grille`. Attention : les numéros de case commencent à 0.

3.6 Exercice 3

Cet exercice est nettement plus difficile, réservé au étudiants ayant déjà bien pratiqué les tableaux et les boucles.

Programmez le jeu de puissance 4 (grille 6 lignes par 7 colonnes). Vous êtes autorisés à utiliser des tableaux et des boucles **`while`** si vous savez vous en servir. Ce n'est pas indispensable.

Le joueur donne le numéro de la colonne et le pion se place à la bonne hauteur.

Le test de victoire n'est pas demandé (mais vous pouvez essayer).

A Rappel sur la boucle `while`

La fonction `main` contient une boucle **`while`**. C'est une notion nouvelle traitée en cours récemment (ou bientôt), voici ce que vous avez besoin de savoir.

Lorsqu'on veut répéter une séquence d'instructions plusieurs fois, on utilise une *boucle*. En C il n'existe qu'une seule sorte de boucle : la boucle **`while`** (toutes les autres sont des variantes de celle-ci). Il est interdit d'utiliser une autre boucle pour l'instant.

La boucle while ressemble à un **`if`** sans **`else`** :

```
while (cond) {
 ... // Corps de la boucle: séquence d'instruction à
     // effectuer ET À RECOMMENCER TANT QUE cond est vraie
}
```

Cette instruction s'exécute de la façon suivante :

1. la condition (`cond`) est évaluée
2. Si elle est fause, on sort immédiatement de l'instruction **`while`** (le corps de la boucle n'est pas exécutée)
3. Si elle est vraie on exécute le corps de la boucle
4. Puis on revient à l'étape 1 ci-dessus.

Notez que si le corps de la boucle ne modifie par la valeur de `cond` le programme va boucler indéfiniment.

Par exemple le programme suivant :

```
int x = 0;
while(x>=0) {
  x = lireInt();
  ecrireString("vous avez tapé: ");
  ecrireInt(x);
  ecriresautDeLigne();
}
```

lit au clavier des nombres jusqu'à ce qu'un nombre négatif soit tapé.