Cours MOOC EPFL d'initiation à la programmation illustré en C++

Quatrième devoir noté
Fonctions

J.-C. Chappelier & J. Sam

1 Exercice 1 — Nombres « lire-et-dire »

1.1 Introduction

Dans cet exercice, on s'intéresse à générer la séquence des nombres « lire-et-dire ». C'est une séquence où l'on applique chaque fois l'opération « lire des chiffres et les dire » pour obtenir le prochain nombre. Concrètement, on « lit à haute voix » (de gauche à droite) le nombre de chiffres consécutifs du nombre précédemment obtenu. C'est plus clair sur des exemples :

- 1 se lit comme « un '1' », donc on obtient 11 ;
- 11 se lit comme « deux '1' », et l'on obtient 21 ;
- 21 se lit « un '2' et un '1' », ce qui donne 1211 ;
- 1211 se lit « un '1', un '2' et deux '1' », ce qui donne 111221 ;
- 111221 se lit « trois '1', deux '2' et un '1' », d'où le nombre suivant 312211 ;
- et ainsi de suite...

Donc, le nombre résultant après avoir appliqué 5 fois l'opération en partant de 1 est 312211. En bref, l'opération donne quelque chose comme

$$N_{\text{même chiffre } c_1 \text{ consécutifs}}\ c_1\ \cdots\ N_{\text{même chiffre } c_k \text{ consécutifs}}\ c_k$$

quand elle est appliqué sur un nombre $c_1 \cdots c_k$, où k est le nombre de *changement* de chiffres. Par exemple pour le nombre 111221 (où $k = 3$, $c_1 = 1$, $c_2 = 2$ et $c_3 = 1$), cela donne :

$$3\ 1\ 2\ 2\ 1\ 1$$

On désire donc écrire un programme qui peut appliquer certain nombre de fois l'opération « lire-et-dire » à partir d'un nombre donné. On se limitera par contre

ici à un petit nombre de répétitions de sorte à pouvoir représenter les nombres obtenus sur un `int` sans problème de débordement de capacité.

Télécharger le programme `lireetdire.cc` fourni sur le site du cours et le compléter suivant les instructions données ci-dessous.

ATTENTION : vous ne devez en aucun cas modifier ni le début ni la fin du programme fourni, juste ajouter vos propres lignes à l'endroit indiqué. Il est donc impératif de respecter la procédure suivante :

1. sauvegarder le fichier téléchargé sous le nom `lireetdire.cc` ou `lireetdire.cpp`;
2. écrire le code à fournir (voir ci-dessous) entre ces deux commentaires :

```
/*************************************************
 * Compléter le code à partir d'ici
 *************************************************/

/*****************************************
 * Ne rien modifier après cette ligne.
 *****************************************/
```

3. sauvegarder et tester son programme pour être sûr(e) qu'il fonctionne correctement, par exemple avec les valeurs utilisées dans l'exemple de déroulement donné plus bas ;
4. soumettre le fichier modifié (toujours `lireetdire.cc` ou `lireetdire.cpp`).

1.2 Méthodologie

La première chose à faire (donnée dans le code fourni : `separer_chiffre_gauche`), est de séparer le chiffre de gauche du reste du nombre pour pourvoir les manipuler. Par exemple pour 1234, récupérer le 1 d'un côté et le 234 de l'autre.

La fonction `separer_chiffre_gauche` fournie prend un nombre en argument et le modifie en supprimant son chiffre le plus à gauche, qu'elle retourne comme valeur de retour. Si `x` vaut `1234`, `separer_chiffre_gauche(x)` vaut `1` et `x` a été modifié en `234`.

Nous aurons aussi besoin des fonctions suivantes :

— (fonction `ajouter_chiffre_droit`) de pouvoir ajouter un chiffre à droite, pour constituer le nouveau nombre ; ceci se fait simplement en multipliant le nombre reçu par 10 et en lui ajoutant le chiffre voulu ; par exemple, si `x` vaut `1234`, `ajouter_chiffre_droit(x, 5)` modifie `x` en `12345` ;

— (fonction `dire_chiffre`) de pouvoir ajouter à droite d'un nombre donné un chiffre et sa répétition, par exemple ajouter « un '1' », c'est-à-dire

« 11 » à droite 3122 (et obtenir donc 312211) ; cela se fait simplement en utilisant deux fois fonction précédente `ajouter_chiffre_droit` ;

— (fonction `lire_et_dire`) de pouvoir appliquer *une* fois l'opération « lire-et-dire » à un nombre donné ; cela se fait en :
  — séparant une première fois le chiffre de gauche (fonction `separer_chiffre_gauche`) ;
  — répétant jusqu'à ce que le nombre manipulé soit nul :
    — séparer à nouveau le chiffre de gauche ;
    — si c'est le même qu'auparavant, augmenter son nombre de répétitions ;
    — sinon ajouter à droite du nombre résultat, le chiffre précédent et son nombre de répétitions (fonction `dire_chiffre`) ;
  — initialisant correctement et mettant correctement à jour les variables dont on a besoin ;
— (fonction `repeter_lire_et_dire`, **fournie**) de pouvoir appliquer *plusieurs* fois l'opération « lire-et-dire » à un nombre donné.

1.3 Exemples de déroulement

Le `main` fourni lit 2 nombres sur l'entrée standard : premièrement le nombre initial et deuxièmement le nombre de fois qu'il faut répéter « lire-et-dire » ; par exemple :

```
1 5
```

Il donne alors le nombre resultant. Pour les données dessus, un programme correct affiche :

```
312211
```

Note : Les données ne vont pas contenir le chiffre 0.

Attention : Quand vous testez votre programme, essayez de ne pas donner de trop grands nombres, comme plus de quelques chiffres pour le nombre initial et plus de 6 répétitions. Si vous ne faites pas attention, vous pouriez dépasser les limites de représentation des `int` et obtenir alors des résultats faux. Nous ne testerons votre code qu'avec des nombres qui respectent ces limites.

2 Exercice 2 — Date de Pâques

Le but de cet exercice est de déterminer la date des Pâques (chrétiennes grégoriennes) : on demande à l'utilisateur d'entrer une année et le programme affiche la date de Pâques de l'année correspondante. Par exemple :

```
Entrez une annee (1583-4000) : 2006
Date de Paques en 2006 : 16 avril
```

On vous demande pour cela d'écrire trois fonctions :

1. une fonction `demander_annee` qui ne prend pas d'argument et retourne un entier ; cette fonction doit :
   — demander une année à l'utilisateur (message : « `Entrez une annee (1583-4000) :` », voir l'exemple d'affichage ci-dessus) ; tant que l'année entrée n'est pas conforme, cette fonction devra reposer la question.
   — vérifier que l'année saisie est bien entre 1583 et 4000 ; sinon redemander ;
   — retourner l'année (correcte) saisie ;
2. une fonction `affiche_date` qui prend deux entiers en paramètres : une année et un nombre de jours compris entre 22 et 56 [1] ; cette fonction doit :
   — afficher le message « `Date de Paques en` » suivi de l'année reçue en premier paramètre, suivi de « : » comme dans l'exemple d'affichage ci-dessus ;
   — si le nombre de jours reçu en second paramètre est inférieur ou égal à 31, afficher le simplement, suivi du mot « mars » ;
   — si ce nombre de jours est supérieur ou égal à 32, lui retrancher 31, et l'afficher suivi du mot « avril » ;
3. une fonction `date_Paques` qui reçoit une année en paramètre (nombre entier) et retourne un entier entre 22 et 56 indiquant le jour suivant la convention appliquée dans la fonction `affiche_date` ; cette fonction doit calculer les valeurs suivantes (il s'agit de l'algorithme de Gauss ; c'est moins compliqué qu'il n'y parait) :
   — le siècle ; il suffit de diviser l'année par 100 ;
   — une valeur `p` qui vaut 13 plus 8 fois le siècle, le tout divisé par 25 ;
   — une valeur `q`, division du siècle par 4 ;
   — une valeur `M` valant `15 - p + siecle - q`, le tout modulo 30 ;
   — une valeur `N` valant `4 + siecle - q` modulo 7 ;
   — une valeur `d` qui vaut `M` plus 19 fois « l'année modulo 19 », le tout modulo 30 ;
   — une valeur `e` qu'il serait trop compliqué de décrire en français et que nous vous donnons directement :

1. Il s'agit du nombres de jours entre Pâques et le dernier jour de Février. La date de Pâques étant toujours comprise entre le 22 mars et le 25 avril, ce nombre sera en fait toujours compris entre 22 et 56 : de 22 à 31 pour un jour de mars et de 32 à 56 pour un jour entre le 1ier et le 25 avril.

`(2 * (annee % 4) + 4 * (annee % 7) + 6 * d + N) % 7`

— le jour (ou presque, voir ci-dessous) : `e` plus `d` plus 22.

Toutes les divisions ci-dessus sont des *divisions entières*, et, pour rappel, « `a` modulo `b` » s'écrit « `a % b` » en C++.

La valeur du jour doit cependant encore être corrigée dans certains cas particuliers :

— si `e` vaut 6

— et que :

- `d` vaut 29
- ou `d` vaut 28 et `11*(M+1)` modulo 30 est inférieur à 19,

alors il faut soustraire 7 au jour.

C'est cette valeur (jour) que devra renvoyer la fonction `date_Paques`.

Complétez ensuite votre programme en utilisant les trois fonctions précédentes dans le `main()` de sorte à avoir le comportement décrit au début de cet exercice.

ATTENTION ! Pour avoir les points, votre programme devra respecter strictement le format d'affichage indiqué au début de l'exercice et ne rien écrire d'autre que la question et sa réponse au format défini, exactement comme dans l'exemple donné plus haut.

Notez qu'il n'y a *pas* d'accents, ni de retour à la ligne après la question. Mais il y a bien *un* retour à la ligne en fin de réponse.

3 Exercice 3 — Mastermind

3.1 Introduction

Le but de cet exercice est de programmer une version texte du jeu de Mastermind. Nous avons voulu ici illustrer l'aspect « modularisation du code » par décomposition de la tâche en de nombreuses fonctions. Plusieurs sont fournies, et vous devez les utiliser ; d'autres seront à écrire.

Télécharger le programme `mastermind.cc` fourni sur le site du cours et le compléter suivant les instructions données ci-dessous.

ATTENTION : vous ne devez en aucun cas modifier ni le début ni la fin du programme fourni, juste ajouter vos propres lignes à l'endroit indiqué. Il est donc impératif de respecter la procédure suivante :

1. sauvegarder le fichier téléchargé sous le nom `mastermind.cc` ou `mastermind.cpp` ;
2. écrire le code à fournir (voir ci-dessous) entre ces deux commentaires :

```
/*****************************************************
 * Compléter le code à partir d'ici
 *****************************************************/

/*****************************************
 * Ne rien modifier après cette ligne.
 *****************************************/
```

3. sauvegarder et tester son programme pour être sûr(e) qu'il fonctionne correctement, par exemple avec les valeurs utilisées dans l'exemple de déroulement donné plus bas ;
4. soumettre le fichier modifié (toujours `mastermind.cc` ou `mastermind.cpp`).

3.2 Code fourni

Le jeu sera constitué d'une combinaison de 4 couleurs (de type `char`) choisies parmi 7 : `'R'`, `'G'`, `'B'`, `'C'`, `'Y'`, `'M'` et `'.'` pour le trou (pas de couleur), qui peut ici être utilisé pour faire une combinaison, donc par exemple : « `M C . G` ».

Comme au niveau de ce devoir vous n'êtes pas encore sensés maîtriser les tableaux en C++, nous manipulerons explicitement les couleurs séparément une à une.

Commencez par prendre connaissance du code fourni, dont les fonctions sont expliquées ici :

— `tirer_couleur` : cette fonction tire une couleur au hasard parmi les couleurs autorisées ;

— `poser_question` : c'est une « fonction outil » que vous n'aurez pas à utiliser directement mais qui est utilisée par `lire_couleur` ; elle est ici pour illustrer l'aspect « modularisation du code » ;

— `lire_couleur` : demande à l'utilisateur d'entrer une couleur, dont elle vérifie la validité grâce à une fonction `couleur_valide` que vous aurez à définir plus bas ;

— `afficher_couleurs` : c'est à nouveau une « fonction outil » que vous n'aurez pas à utiliser directement ; elle affiche les 4 couleurs du jeu ou d'une proposition du joueur ;

— `afficher` : cette fonction permet d'afficher `nb` fois le caractère `c` ; vous devrez l'utiliser pour afficher les réponses à donner au joueur ; en effet, *vous n'avez pas le droit de faire appel directement à* `cout` *dans votre code* ;

— `afficher_coup` : affiche un coup du jeu en affichant les 4 couleurs proposées par le joueur et la réponse correspondante ; vous devrez utiliser cette fonction dans la fonction principale qui gère le jeu (fonction `jouer`, voir plus bas) ; la fonction `afficher_coup` reçoit 4 couleurs en arguments, les 4 couleurs proposées par le joueur, et 4 caractères pour représenter la réponse ; ce dernier point sera expliqué plus bas ;

— `message_gagne` : affiche le message à donner au joueur lorsqu'il a gagné ; vous devrez utiliser cette fonction dans la fonction principale qui gère le jeu ; cette fonction reçoit le nombre de coups joués ;

— `message_perdu` : affiche le message à donner au joueur lorsqu'il a perdu ; vous devrez utiliser cette fonction dans la fonction principale qui gère le jeu car, nous répétons, *vous n'avez pas le droit de faire appel directement à* `cout` *dans votre code* ; cette fonction reçoit les 4 couleurs du jeu (celles qu'il fallait trouver, donc).

3.3 Code à fournir

On vous demande dans cet exercice de fournir 6 fonctions, dont certaines très simples (1 ligne). Certains prototypes sont déjà fournis, d'autres sont à écrire.

— `couleur_valide` : détermine si le caractère reçu en paramètre est une couleur, c.-à-d. s'il est soit `'.'`, `'R'`, `'G'`, `'B'`, `'C'`, `'Y'` ou `'M'` ;

— `verifier` : vérifie si la couleur reçue (type `char`) en premier paramètre correspond à celle donnée en second paramètre, et si c'est le cas cette fonction augmente de 1 le score reçu en troisième paramètre ;
vous devez également faire en sorte que le second paramètre soit modifié en quelque chose qui **n'**est **pas** une couleur (par exemple `'x'`) de sorte que lorsque l'on utilise cette fonction, l'on sache que la couleur testée (ce second paramètre, justement) a déjà été traitée positivement ; on ne peut en effet pas avoir plusieurs réponses (« bonne couleur » ou « bonne couleur bien placée ») pour une même couleur de référence (couleur à deviner) ; lors du calcul de la réponse, il faut donc marquer toute couleur de référence déjà traitée ;
enfin, la fonction `verifier` retourne « `true` » si la couleur reçue en premier paramètre correspond à celle donnée en second paramètre, et « `false` » dans le cas contraire ;

— `apparier` : cette fonction sert à tester une couleur proposée par rapport aux trois couleurs de référence situées dans des autres colonnes (couleur

proposée mal placée) ; cette fonction doit recevoir :
— une couleur à tester (couleur proposée) ;
— trois couleurs candidates à tester (couleurs de référence) ; ces trois couleurs doivent pouvoir être modifiées par la fonction ;
— un nombre, à modifier, qui sera augmenté de 1 si la couleur proposée à tester correspond à l'une des trois couleurs de référence ;

cette fonction doit simplement appeler `verifier` avec la couleur proposée à tester et, tour à tour, chacune des trois couleurs de référence ; notez cependant bien que l'on ne teste la seconde couleur de référence que si la couleur proposée à tester ne correspondait pas à la *première* couleur de référence, et l'on ne teste la troisième couleur de référence que si aucune des deux premières ne correspondait à la couleur proposée à tester ;

— `afficher_reponses` : cette fonction est peut-être la plus complexe ; elle construit la réponse à apporter au joueur lorsqu'il a proposé 4 couleurs ; elle reçoit en paramètres (dans cet ordre) : les 4 couleurs proposées à tester et les 4 couleurs de référence (dans le même ordre de colonnes que celles à tester) ;

en utilisant la fonction `afficher`, la fonction `afficher_reponses` doit afficher autant de `'#'` que de couleurs proposées correctes bien placées (cela correspond aux petits pions noirs ou rouges dans les versions classiques du Mastermind), ***puis*** autant de `'+'` que de couleurs proposées présentes dans la combinaison du jeu mais mal placées (cela correspond aux petits pions blancs ou jaunes dans les versions classiques du Mastermind) et enfin autant de `'-'` que de couleurs proposées fausses ;

par exemple, si le jeu est « `M..R` », et que le joueur propose « `CYM.` », il faudra répondre « `++--` » (le `'M'` et un des trous (`'.'`) sont effectivement dans le jeu, mais mal placés dans la proposition du joueur) ; et si le jeu est « `RRGG` », et que le joueur propose « `YGGR` », il faudra répondre « `#++-` » (la troisième couleur proposée, `'G'`, est bien placée, et il y a deux autres couleurs proposées qui sont justes, mais mal placées, le `'R'` et l'autre `'G'`) ;

pour construire la réponse, procéder comme suit :
— tester une à une chacune des couleurs pour voir si elle est bien placée ;
— *ensuite*, une fois ces 4 tests effectués, pour chacune des couleurs proposées qui n'étaient pas bien placées, chercher, à l'aide de la fonction `apparier`, si elle s'apparie avec une des trois autres couleurs de référence (que celle à la même position) ;
— afficher enfin le reste de `'-'` pour toutes les mauvaises couleurs ;

— `gagne` : cette fonction très simple retourne simplement « `true` » si la

combinaison de couleurs proposées, donnée par ses 4 premiers arguments, correspond exactement à celle recherchée, indiquée par ses 4 derniers arguments ; et sinon, elle retourne « `false` » ;

— `jouer` : le cœur du jeu, combinant les autres fonctions écrites ;
cette fonction devra recevoir un paramètre *optionnel* indiquant le nombre de coups maximum autorisés pour le joueur ; par défaut, le nombre maximum de coups sera 8 ;
le déroulement de la fonction `jouer` est le suivant :
  — commencer par tirer 4 couleurs au hasard (à l'aide de `tirer_couleur`) ; cela constitue la combinaison de référence, à deviner ;
  — ensuite, tant que le joueur n'a pas gagné (fonction `gagne`) et que le nombre de coups est inférieur ou égal au nombre maximum de coups :
    — demander 4 fois une couleur (fonction `lire_couleur`) ;
    — puis afficher le coup et sa réponse (fonction `afficher_coup`) ;
  — à la fin afficher le message adéquat : celui de `message_gagne` si le joueur a trouvé, et sinon, celui de `message_perdu`.

3.4 Exemples de déroulement

Voici 2 exemples de déroulement, l'un gagnant, l'autre perdant. Pour simplifier nous avons ici supprimé la saisie des coups par le joueur (sauf le premier).

```
Entrez une couleur : r
'r' n'est pas une couleur valide.
Les couleurs possibles sont : ., R, G, B, C, Y ou M.
Entrez une couleur : R
Entrez une couleur : R
Entrez une couleur : G
Entrez une couleur : G
 R R G G : +---
Entrez une couleur : [...] BCYM
 B C Y M : #+--
Entrez une couleur : [...] .RBC
 . R B C : ++--
Entrez une couleur : [...] GYM.
 G Y M . : +---
Entrez une couleur : [...] YBYR
 Y B Y R : ####
Bravo ! Vous avez trouvé en 5 coups.

Entrez une couleur : R
```

```
Entrez une couleur : R
Entrez une couleur : R
Entrez une couleur : R
 R R R R : #---
Entrez une couleur : [...] GGGG
 G G G G : ----
Entrez une couleur : [...] BBBB
 B B B B : ----
Entrez une couleur : [...] ....
 . . . . : #---
Entrez une couleur : [...] CCCC
 C C C C : ----
Entrez une couleur : [...] YYYY
 Y Y Y Y : #---
Entrez une couleur : [...] MY.R
 M Y . R : ##++
Entrez une couleur : [...] MYR.
 M Y R . : #+++
Perdu :-(
La bonne combinaison était :  M . Y R
```