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

Fourth assignment
Functions

J.-C. Chappelier & J. Sam

1 Exercise 1 — Look-and-Say Numbers

1.1 Introduction

In this exercise, we want to generate sequences of "look-and-say numbers". These are sequences where, each time, we apply the "look-and-say" (="read aloud") operation to obtain the next number. The "look-and-say" operation consists in "reading out" (from left to right) the sequences of digits of the number. It's better explained with some examples:

- 1 is read as "One 1", therefore it becomes 11;
- 11 is read as "Two 1s", therefore it becomes 21;
- 21 is read as "One 2 and one 1", therefore it becomes 1211;
- 1211 is read as "One 1, one 2 and two 1s", therefore it becomes 111221;
- 111221 is read as "Three 1s, two 2s and one 1", therefore it becomes 312211;
- and so on...

Therefore, the resulting number of applying 5 times the look-and-say property to 1 is 312211. In short, this property gives us something like

$$N_{\text{consecutive same digit } d_1}\ d_1\ \cdots\ N_{\text{consecutive same digit } d_k}\ d_k$$

when operating on a number $d_1 \cdots d_k$, where k is the number of digit *changes*. For instance, with the number 111221 (where $k = 3$, $d_1 = 1$, $d_2 = 2$ and $d_3 = 1$), we have:

3 1 2 2 1 1

We would like to write a program that can apply, a given number of time, the "look-and-say" operation to some given number. In this assignment, we will limit our tests to a very few number of repetitions, on small numbers, so as not to overflow the `int`-type representation capacity.

Download the source code available at the course webpage and complete it according to the instructions below.

WARNING: you should modify neither the beginning nor the end of the provided file. It's thus mandatory to proceed as follows:

1. save the downloaded file as `lireetdire.cc` or `lireetdire.cpp`;
2. write your code between these two provided comments:

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

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

3. save and test your program to be sure that it works properly; try for instance the values used in the example given below;
4. upload the modified file (still named `lireetdire.cc` or `lireetdire.cpp`) in "OUTPUT submission" (not in "Additional"!).

1.2 Methodology

First, we need to be able to operate on the digits of a number from left to right in order to be able to "read it out loud". For this, we need a function to get us the leftmost digit of a number and remove it from that number so that we can go on, operating on the next digit.

The provided function `separer_chiffre_gauche` takes a number and modifies it, removing its left-most digit which is returned (as return value). If some variable `x` is `1234`, `separer_chiffre_gauche(x)` returns `1`, and `x` has been modified to `234`.

When we have this, we need some other functions to apply get the "look-and-say" sequence from a given number:

- (function `ajouter_chiffre_droit`, which means "add/push right digit") so as to create the new number in the sequence, we need to be able to add a digit to the right of some number; for instance, if `x` is `1234`, then `ajouter_chiffre_droit(x, 5)` modifies `x` into `12345`;
- (function `dire_chiffre`, which means "say digit") we also need to be able to add to the right of some number, a digit as well as its number of occurrences; for instance add « One '1" », i.e. « 11 », to the right of 3122 (thus getting 312211); this can easily be achieved with two calls to the former function `ajouter_chiffre_droit`;
- (function `lire_et_dire`, which means "look/read and say") we furthermore need to be able to apply *once* the "look-and-say" operation to some given number; this can be done by:
  - separate a first time the left digit (fonction `separer_chiffre_gauche`);
  - repeat until the manipulated number is null:
    - separate once more the left digit;
    - if it's the same as before, increase its number of repetitions;
    - otherwise add to the result, the preceeding digit with its number of repetitions (fonction `dire_chiffre`);
  - properly initialize and properly update all the intermediate variables needed;
- (function `repeter_lire_et_dire`, **provided**, it means "repeat look/read and say") finaly, we need to apply *several times* the "look-and-say" operation to some given number.

1.3 Execution examples

Your program will take 2 numbers from standard input: the first number and the number of times to apply the property, such as

```
1  5
```

and output the resulting number. For the above input, the correct program would output the following:

```
312211
```

Notice: There will be no 0 digits in the input.

Attention: When testing, try not to provide too large numbers as input, such as more than one digit to the first number and more than 6 as the number of times. If you don't pay attention to this, you will exceed the inherent limits of your computer. We will make sure to test your code with numbers that fit into these limits.

2 Exercise 2 — Easter's date

The aim of this exercise is to determine (Christian Gregorian) Easters' dates: we will ask the user to enter a year and the program will display the easter's date of the corresponding year. For example ("Pâques" means Easter in French):

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

In order to do so, you are asked to write three functions:

1. a function `demander_annee` which does not have any argument and return a integer; this function should;
   - ask a year to the user (message:
     "`Entrez une annee (1583-4000) :  `",
     see the display example given in the procedure example above;
   - verify that the entered year is in between 1583 and 4000; otherwise ask again;
   - return the entered year (when that year is correct);
2. a function `affiche_date` which takes two integers as parameters: a year and a number in between 22 and 56 [1]; this function has to:
   - display the message "`Date de Paques en`", followed by the year given as the first parameter, and then a colon ": ", as shown in the procedure example above;
   - if the number given in the second parameter is smaller or equal to 31 display this number followed by the word "mars" (means "March");

[1] it represents the number of days in between easter and the last day of February. Because Easter always takes place in between the 22nd of march and the 25th of april, this number will always be in between 22 and 56: from 22 to 31 for a march day and from 32 to 56 for a day in between the 1st and the 25th of april.

- if this number is bigger or equal to 32, subtract 31 and then display it followed by the word “avril” (means “April”);

3. a function `date_Paques` which receives a year as a parameter (integer) and returns a integer between 22 and 56, which indicates the date using the convetion used by the `affiche_date` function; the `date_Paques` function should calculate the following values (this is the Gauss algorithm; it is less complicated than it seems at first);

   - the century: you only have to divide the year by 100;
   - a value `p` that equal 13 plus 8 times the century, and then the whole divided by 25;
   - a value `q`, which is the century divided by 4;
   - a value `M`, as `15 - p + siecle - q`, the whole modulo 30;
   - a value `N`, as (`4 + siecle - q`) modulo 7;
   - a value `d` equal to `M` plus 19 times “the year modulo 19”, and then the whole thing modulo 30;
   - and a value `e` which is too hard to explain and that we are giving to you directly:

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

   - the day (except for a few exceptions, see the note below): `e` plus `d` plus 22.

   All the above divisions are *division of integers*, furthermore as a reminder: “`a` modulo `b`” is written “`a % b`” in C++.

   The day value needs to be corrected in special cases:

   - if `e` equals 6
   - and:
     - `d` equals 29
     - or `d` equals 28 and `11*(M+1)` modulo 30 is smaller than 19,

   in these cases you need to subtract 7 to the day.

   It is this value (day) that the function `date_Paques` should return.

Once the function written, complete your program by using these three functions in the `main()` in order to have the right program behaviour described at the beginning of this exercise.

ATTENTION! In order to have the maximum grade, the program must strictly respect the display format given at the beginning of the exercise, nothing else!

Note that there are *no* diacritics in, nor endline after the question. However there is *one* endline at the end of the answer.

3 Exercise 3 — Mastermind

3.1 Introduction

The aim of this exercise is to code a text version of the Mastermind game. In this exercise, we want to illustrate the aspect of "code modularity" by separating the task into multiple very specific functions. Some of these functions are provided, and you must use them; others must be written.

Download the source code available at the course webpage and complete it according to the instructions below.

WARNING: you should modify neither the beginning nor the end of the provided file. It's thus mandatory to proceed as follows:

1. save the downloaded file as `mastermind.cc` or `mastermind.cpp`;
2. write your code between these two provided comments:

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

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

3. save and test your program to be sure that it works properly; try for instance the values used in the example given below;
4. upload the modified file (still named `mastermind.cc` or `mastermind.cpp`) in "OUTPUT submission" (not in "Additional"!).

3.2 Given code

The aim of the game will be to guess a combination of 4 colors (`char` type) among the following 7: `'R'`, `'G'`, `'B'`, `'C'`, `'Y'`, `'M'` and `'.'` for the hole

(no color), which can here be used in a combination like a color, for example: "`M C . G`".

We will treat each color separately in the code. Indeed, regarding the course schedule, you are not supposed to know how to use arrays/vectors yet.

Start by looking at and understanding the given code. Each function is explained below:

- `tirer_couleur` (means "draw color" [at random]): this function draws a random color among the available colors;

- `poser_question` (means "ask question"): this is a "tool function" that you will not have to use directly. It is already used in `lire_couleur`; the aim of having this function in this exercise is too illustrate "code modularity";

- `lire_couleur` (means "read color" [from standard input]): asks the user to enter a color and then verifies whether this color is valid or not by using the function `couleur_valide` that you have to write (see below);

- `afficher_couleurs` (means "display colors"): this is again a "tool function" that you will not have to use directly; it displays 4 colors that can be either the 4 reference colors or the user's guess;

- `afficher` (means "display/print"): this function allows you to display `nb` times the character `c`; you shall use it to display the feedback given to the user about his guess; indeed, *you are forbidden in this exercise to call* `cout` *directly in your code*;

- `afficher_coup` (means "display move/step/guess"): displays one game step by displaying both the 4 colors proposed by the player and the corresponding feedback; you should use this function in the function that is managing the game (function `jouer`, see below); the `afficher_coup` function receives 4 colors as arguments, corresponding to the 4 colors guessed by the player, and 4 characters that are the feedback given to the player; this feedback is explained below;

- `message_gagne` (means "wining message"): displays the message to be given when the player wins; you shall use this function in the main function that is managing the game; this function receives the number of game guesses already played as an argument;

- `message_perdu` (means "loosing message"): displays a message when the player lost; you should use this function in the main function that is managing the game. Indeed, we are repeating it, *you are forbidden to call directly* `cout` *in your code in this exercise*; this function should receive the 4 reference colors combination as an argument (so the right combination that the player was trying to find).

3.3 Code to be written

In this exercise you are asked to write 6 functions. Some are really simple (1 line). Some prototypes are already provided, others must be completed.

- `couleur_valide` (means "valid color"): verifies that the character received as a parameter is a color, which means one of the following `'.'`, `'R'`,`'G'`,`'B'`,`'C'`,`'Y'` or `'M'`;

- `verifier` (means "check"): checks whether the color (type `char`) received as the first parameter corresponds to the one given as the second parameter, or not; if it is the case, the score (which is the third parameter) should be increased by 1;

  you should also modify the second parameter into a character that is **not** a color (for example `'x'`); the aim of this manipulation is to mark that the tested color (the second parameter) has already been treated; indeed we can't have for the same guess several feedbacks (« good color » or « good color at the right place ») for the same color; so we should mark somehow that this color has already been handled;

  the function `verifier` returns « `true` » if the color given as the first parameter corresponds to the one given as the second parameter, and « `false` » if it is not the case;

- `apparier` (means "match"): this function tests if a guessed color corresponds to one of the three reference colors that are present but are not aligned (checks if the guessed color is good but not correctly placed); it should receive as arguments

  - a candidate color to test;
  - three reference colors that should be tested; the function should be able to modify these three colors;
  - a number, to be modified, which will be increased by 1 if the candidate color matches one of the three reference colors;

- `afficher_reponses` (means "display feedback"): this function might be the most complex one; it builds the feedback for the player once he has made a 4 color guess;

  the function takes as arguments (in this order): the 4 colors to be tested and the 4 reference colors (in the same column order as the colors to test);

  using the function `afficher`, it displays as many `'#'` as good and correctly placed colors (this `'#'`-sign corresponds to the number of black or red markers in the classic Mastermind game), ***then*** as many `'+'` as there are good yet misplaced colors (this `'+'`-sign corresponds to the white or yellow markers in the classic Mastermind game) and, finally, as many `'-'` as there are wrongly guessed colors;

  for example, if the reference is "`M..R`" and the player's guess is "`CYM.`" it should answer "`++--`" (the `'M'` and one of the holes (`'.'`) are in the reference combination, but misplaced); if the reference is "`RRGG`", and the player guesses "`YGGR`", it should answer "`\#++-`": the third color, `'G'` is correct and well placed, and there are two other colors that are in the reference combination but misplaced (the `'R'` and the other `'G'`);

  to build the feedback, proceed as follows:

  - test one by one each guessed color to see if they are correctly placed;
  - *then*, once these 4 tests have been performed, search for each of the guessed colors that was not correctly placed if it corresponds to one of the other three reference colors (i.e. the three colors that are at a different position); to do so, use the function `apparier`);
  - finally display the remaining `'-'` for all the wrongly guessed colors;

- `gagne` (means "wins/won"): this very simple function simply returns « `true` » if the color combination given in the 4 first arguments exactly corresponds, one by one, to the 4 last arguments; otherwise, it return « `false` »;

- `jouer` (means "play"): the core of the game which combines all the previously written functions;

  this function should receive an *optional* parameter indicating the maximum number of guesses the player is allowed to make; the default maximum number is 8;

  it proceeds as follows:

  - start by drawing 4 random colors (with `tirer_couleur`), this will be the reference combination to be guessed;

- then, as long as the player hasn't won yet (function `gagne`) and the number of guesses is smaller or equal to the maximum number of guesses allowed:
  - ask for 4 colors (4 times function `lire_couleur`);
  - then display the guessed combination and its corresponding feedback (function `afficher_coup`);
- at the end, display the appropriate message: the `message_gagne` if the player won or the `message_perdu` if he lost.

3.4 Execution examples

Here are 2 execution examples, one where the player wins, the other where he loses. To simplify the display, we removed the questions displayed to the player to enter the colors (except for the first move).

```
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
```