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

Quatrième devoir noté
Fonctions

V. Lepetit, J.-C. Chappelier & J. Sam

du 18 octobre au 04 novembre 2013

1 Exercice 1 – Sommes et produits

On s'intéresse ici à trouver des nombres dont la somme des chiffres est égale à leur produit. Par exemple 123 est un tel nombre : $1 + 2 + 3 = 1 \times 2 \times 3$.

On commence par vouloir écrire une fonction qui retourne la somme des chiffres d'un nombre passé en paramètre. Par exemple, cette fonction devra retourner 3 quand 12 est passé en paramètre (puisque $1 + 2 = 3$) et 7 quand 223 est passé en paramètre ($2 + 2 + 3 = 7$).

Mais comment calculer la somme des chiffres d'un nombre n ?

– Tout d'abord, on peut remarquer que le reste de la division de n par 10 (c'est-à-dire « n modulo 10 », noté en C++ `n % 10`), donne le chiffre le plus à droite de n. Par exemple, `178 % 10` donne `8`.
– Pour obtenir le deuxième chiffre le plus à droite de n, on commence par diviser n par 10, en utilisant la division *entière*. Avec $n = 178$, on obtient 17. Il suffit alors de prendre ce nouveau nombre modulo 10 pour obtenir le deuxième chiffre. Par exemple, `17 % 10` nous donne `7`.
– Il n'y a plus qu'à répéter ces opérations jusqu'à ce que le résultat soit 0. En continuant l'exemple précédent : 17 divisé par 10 donne 1 (division entière), 1 modulo 10 donne 1, 1 divisé par 10 donne 0, on s'arrête.
– En sommant les chiffres obtenus grâce au modulo 10 au fur et à mesure qu'on les calcule, on obtient la somme des chiffres du nombre n de départ.
  Pour résumer, avec $n = 178$ cela donnera les opérations suivantes :
  · `178 % 10` donne `8` ;
  · `178 / 10` donne `17` ;

- `17 % 10` donne `7`, que l'on ajoute au `8` précédent ; cela fait `15` ;
- `17 / 10` donne `1` ;
- `1 % 10` donne `1`, que l'on ajoute au `15` précédent ; cela fait `16` ;
- `1 / 10` donne `0` ; on s'arrête, le résultat est `16`.

Ecrivez une fonction appelée `somme` qui, en appliquant la méthode décrite ci-dessus, retourne la somme des chiffres du nombre entier qu'elle reçoit en paramètre. La valeur de retour est donc également un nombre entier.

Ecrivez ensuite une fonction appelée `produit` qui retourne le produit des chiffres d'un nombre passé en paramètre. Par exemple, cette fonction devra retourner 2 quand 12 est passé en paramètre ($1 \times 2 = 2$), 24 quand 423 est passé en paramètre ($4 \times 2 \times 3 = 24$). Le principe de calcul est le même que pour la somme.

Utilisez ensuite vos deux fonctions pour écrire une fonction `somme_produit_egaux` qui teste si la somme des chiffres d'un nombre entier passé en paramètre est égale au produit de ces mêmes chiffres. Par exemple, `somme_produit_egaux(12)` devra renvoyer `false`, `somme_produit_egaux(123)` devra renvoyer `true`.

Note : on peut écrire cette fonction en une seule ligne.

Utilisez enfin cette fonction `somme_produit_egaux` pour afficher les 20 premiers nombres entiers plus grands que 10 dont la somme des chiffres est égale au produit de ces mêmes chiffres. Pour vous aider à vérifier votre programme, voici les 5 premiers nombres que vous devriez trouver : 22, 123, 132, 213 et 231.

ATTENTION ! Pour avoir les points, votre programme devra respecter strictement le format d'affichage suivant : liste de nombres en une seule ligne, séparés par des virgules. Pour faciliter l'écriture de votre code le dernier nombre sera également suivi d'une virgule. Il n'y aura qu'un seul retour à la ligne, à la fin. Avec 5 nombres, cela donnerait :

```
22, 123, 132, 213, 231,
```

Notez bien qu'il y a une virgule (et une [1] espace) à la fin.

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 :

1. L'espace du typographe est féminine.

```
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 [2] ; 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` 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 :

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

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

2. 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 1er et le 25 avril.

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`.

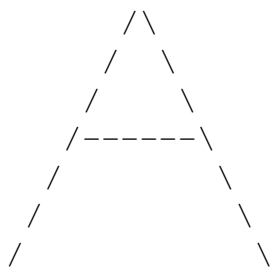
Complétez ensuite votre programme en utilisant les trois fonctions précédentes dans le `main()` de sorte à avoir le comportement décrit au début de cet exercice.

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 – Dessins de lettres

Le but de cet exercice est d'écrire, à partir de fonctions prédéfinies à réutiliser, plusieurs fonctions permettant d'afficher de grands caractères, comme par exemple :



Pour simplifier, on supposera ici ainsi que dans toute la suite que la hauteur demandée pour ces caractères est toujours impaire et supérieure ou égale à 3.

Télécharger le programme `lettres.cc` fourni [3] et le compléter.

3. Adresse : `https://d396qusza40orc.cloudfront.net/intro-cpp-fr/assignments-data/lettres.cc`

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

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

```
/*****************************************
 * Completez le programme a partir d'ici.
 *****************************************/

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

3. sauvegarder et tester son programme pour être sûr(e) qu'il fonctionne correctement, par exemple avec les valeurs données plus bas ;
4. rendre le fichier modifié (toujours `lettres.cc` ou `lettres.cpp`) dans « OUTPUT submission » (et non pas dans «Additional»).

Dans le code téléchargé, vous trouverez les fonctions suivantes :

```
void espaces(int decalage)
{
  for(int i(0); i < decalage; ++i) {
    cout << ' ';
  }
}

void barre_horizontale(int longueur, int decalage)
{
  espaces(decalage);
  for(int i(0); i < longueur; ++i) {
    cout << '-';
  }
}

void barre_verticale(int decalage)
{
  espaces(decalage);
  cout << '|';
}

void barre_a_droite(int decalage)
```

```
{
  espaces(decalage);
  cout << '/';
}

void barre_a_gauche(int decalage)
{
  espaces(decalage);
  /* Attention, cette ligne affiche bien UN SEUL caractere \ !  *
   * En C++, il faut repeter deux fois le caractere \ pour en   *
   * n'afficher qu'un seul.                                     */
  cout << '\\';
}

void fin_ligne()
{
  cout << endl;
}

void affiche_N(int hauteur)
{
  for(int i(0); i < hauteur; ++i) {
    barre_verticale(0);
    barre_a_gauche(i);
    barre_verticale(hauteur-i-1);
    fin_ligne();
  }
}
```

Regardez ce code fourni pour comprendre ce qui vous est donné et que vous *devez* réutiliser.

Vous n'avez pas le droit d'utiliser `cout` pour écrire les fonctions qui vous sont demandées ci-dessous. Vous devez en revanche utiliser des fonctions parmi celles fournies ci-desssus : `barre_horizontale`, `barre_verticale`, `barre_a_droite`, `barre_a_gauche` et `fin_ligne`.

La fonction `affiche_N` ci-dessus est un exemple d'utilisation de ces fonctions.

Notez bien que vous n'avez pas le droit d'appeler directement la fonction `espaces`.

Ecrivez la fonction d'en-tête :

```
void affiche_L(int hauteur)
```

qui permet d'afficher un L sur un nombre de lignes défini par le paramètre `hauteur`. Par exemple, l'appel `affiche_L(7)` devra afficher :

```
|
|
|
|
|
|
-------
```

ATTENTION ! Vous ne devez pas afficher d'espace supplémentaire sinon votre code sera considéré comme faux. Par exemple, dans le dessin du L ci-dessus il n'y a *aucune* espace.

Ecrivez ensuite la fonction d'en-tête :

```
void affiche_O(int hauteur)
```

qui permet d'afficher un O sur un nombre de lignes défini par le paramètre `hauteur`. Par exemple, l'appel `affiche_O(7)` devra afficher la figure de gauche ci-dessous :

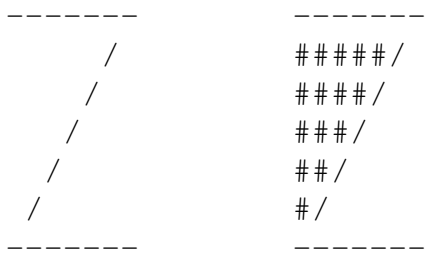
```
 -----          #-----
|     |         |#####|
|     |         |#####|
|     |         |#####|
|     |         |#####|
|     |         |#####|
 -----          #-----
```

La figure de droite (que votre programme ne devra PAS afficher) vous montre les espaces de la figure de gauche en les remplaçant par des `#`, afin que vous voyez exactement où il faut en mettre.

Ecrivez la fonction d'en-tête :

```
void affiche_Z(int hauteur)
```

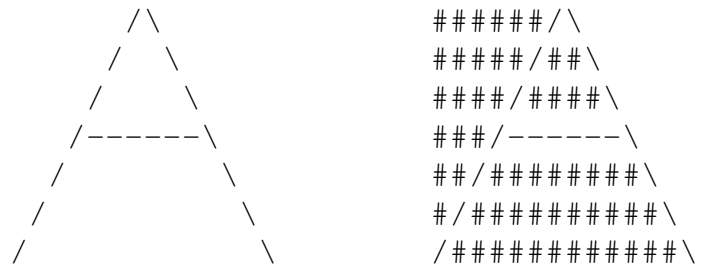
qui permet d'afficher un Z sur un nombre de lignes défini par le paramètre `hauteur`. Par exemple, l'appel `affiche_Z(7)` devra afficher (le dessin de gauche uniquement. Le dessin de droite n'est fourni ici que pour indication et ne devra pas être produit.) :



Ecrivez la fonction d'en-tête :

```
void affiche_A(int hauteur)
```

qui permet d'afficher un A sur un nombre de lignes défini par le paramètre `hauteur`. Par exemple, l'appel `affiche_A(7)` devra afficher (le dessin de gauche uniquement. Le dessin de droite n'est fourni ici que pour indication et ne devra pas être produit.) :



Faites également bien attention à ne pas ajouter de ligne vide supplémentaire en fin de «dessin» de lettre : seules les lettres doivent être dessinées par ces fonctions, sans aucune ligne supplémentaire même vide (mais il y a bien un retour à la ligne en fin de lettre).