Série 5: Fonction (1)

Portée, paramètres, conception

Lien avec le MOOC Initiation à la Programmation (en C++)
Lien avec ICC-Théorie en complément du MOOC

Les éléments du MOOC sur les fonctions sont répartis sur deux semaines. La première partie se concentre sur la conception de fonctions en insistant sur la maîtrise des concepts de portée et de communication entre la fonction et le reste du programme à l'aide de paramètres et de l'instruction return.

La semaine prochaine traitera le sujet de la conception de fonctions récursives. On fournira également quelques outils de mesure du coût calcul effectif pour une exécution donnée d'un exécutable.

Un exercice complémentaire en prévision du projet est fourni en page 2.

Exercices partiels semaine4 du MOOC

- **Document Tutoriel 1ière partie seulement « Calcul de moyenne »**
  - Écriture d'une fonction passant 2 valeurs et retournant leur moyenne

- **Document Exercices semaine 4 du MOOC : 1ière sélection**

  - **Exercice 11 : prototypes (niveau 1, niveau 2 pour questions 4 et 5)**
    - Illustre les bonnes pratiques de vérification des entrées

  - **Exercice 12 : passage de paramètres (niveau 1)**
    - Question : faut-il utiliser le passage par valeur ou par référence ?

  - **Exercice 13 : la fonction cosinus (niveau2)**
    - Décomposition d'un problème en sous-problèmes
    - Chaque sous-problème est traité à l'aide d'une fonction

- **Document Exercices additionnels semaine 4 du MOOC : 1ière sélection**

  - **Exercice 7 : fonctions simples (niveau 1)**
    - Quelle est la bonne pratique à mettre en œuvre concernant les paramètres ?

  - **Exercice 8: Sapin (niveau 2)**
    - Mise en œuvre des grands principes : Abstraction et Ré-utilisation
    - Entrainement pour la décomposition d'un problème

Complément en prévision du Projet : ImageMagick

Découverte des outils de manipulation d'image en mode ligne de commande (in english)

In this exercise we will learn how to use a third-party-application from the command line. We will do it by using the image manipulation tool called ImageMagick. We will use three of the tools it provides: **display, identify** and **convert.** However; note that ImageMagick has a lot more features than the ones presented here.

- **Displaying an image**
  - Create a new directory **`project`** in your directory **`programmation`**.
  - Open a terminal and go into your **`project`** directory with the **`cd`** command.
  - Download the archive file with 2 images in png format in this project directory and "**extract all**".
  - Enter the command **`display epfl.png`** or double click on the file to see the image.
    Notice that you have to close the window displaying the image if you want to continue using the command line of the terminal… If you don't want to block the command line while still viewing the image, simply add the character **&** at the end of the command (like in série1), like this: **display epfl.png &**

  - **What if the image is small ?** You may use the menu proposed by the window displaying the image *but* this has the undesired side effect of interpolating the pixel values, resulting in a blurry image. Read below the **convert** command for scaling an image without such interpolation.

One problem we have with the png format is that such a file cannot be opened and edited with our usual program editor geany because it is not encoded in alphanumeric characters such as the ASCII code. For this reason we need to convert such a format into the PGM format that can be opened in a standard text editor.

- **Converting image formats and scaling with no interpolation**
  - Let's produce a plain, readable by human, image file where we can read the pixel values in geany. To do this, we should use the **`convert`** command with the **-compress none** option:
    `convert -compress none epfl.png epfl.pgm`
  - Display the *text content* of the resulting image with the **cat**[1] command: **`cat epfl.pgm`**
    Each integer is the grey level intensity of one pixel coded with one byte => values are within [0, 255].
  - Now let's scale the resulting pgm file to get a more visible version without the undesired interpolation that produce a blurry image. Here we provide a different file name for the result :
    `convert epfl.pgm –scale 800% epfl_big.pgm`

The project will use the bitmap **pbm** format as input; pixels can only be black (value 1) or white (value 0). You will be able to display and scale such images as shown above. In addition you may want to compare two images used for testing the program. Scale both images before comparing them if they are small.

- **Comparing two images**
  Both of the following syntax of the **compare** command will highlight **in magenta** any pixel that is different in the 2 images ; this is why the output file **difference.png** is in png format ; here we compare the 2 provided images after converting them in the pbm format (you can open them in geany too to notice that it's not easy to spot the differences between the 2 images in a text file format) :

```
convert -compress none epfl.png  epfl.pbm
convert -compress none epfl2.png epfl2.pbm

compare epfl.pbm     epfl2.pbm     difference.png
```

  This variant immediately displays the difference between epfl.pbm and epfl2.pbm :

```
compare epfl.pbm     epfl2.pbm     miff:-  ¦ display
(Note: if you're using a laptop, the key ¦ equals to | )
```

[1] cat is standard LINUX command line tool to concatenate and list files. It is also used to display the text file. For its detailed use check its manual page "man cat"