Langage C – PEIP2 A

Introduction

Exercice 0 – Find my number

1. Create a new file with code blocks or onlinegdb

2. Replace **"Hello World"** by **"\tWelcome PeiP2 A to your First Basic Game :\n\tThe Secret Value\n\tThe rule :You will have to find a secret integer\n \tbelonging to [0-99] in a minimal number of run.\n\n"**

3. Test Your program, you should have the following on your screen :

```
Welcome PeiP2 A to your First Basic Game :
The Secret Value
The rule :You will have to find a secret integer
belonging to [0-99] in a minimal number of run.
```

4. Create a variable **secretValue** of type integer.

5. Create a variable **myAnswer** of type integer.

6. Create a variable **runNumber** of type integer, initialize this variable to the value 0.

7. Create a const variable **maxRunNumber** of type integer which takes the value 5.

8. Initialize **secretValue** with a random value in [0 ; 99] (CM_langage_C_0_toolbox.pdf)

9. Clean the screen (CM_langage_C_0_toolbox.pdf)

10. Print the value of **secretValue** on the screen (CM_langage_C_1_1_variables.pdf)

11. Increase **runNumber** of 1.

12. Print an invitation to the player :

```
You have 5 tries to find the secretValue belonging to [0-99], give your try
number 1 :
```

The 5 will be the const variable **maxRunNumber** and the 1 is the int variable **runNumber**.

13. Then get the value from the keyboard thanks to the function scanf (CM_langage_C_1_1_variables.pdf) and put it in the variable **myAnswer**. After the scanf call the given function my_fflush (CM_langage_C_0_toolbox.pdf) in order to clean the keyboard.

14. Compare **myAnswer** and the **secretValue**, (CM_langage_C_1_2_conditions_et_boucles.pdf).

if **myAnswer** is smaller than the **secretValue** print :

```
The secretValue is greater than ???
```

where ??? is the value of MyAnswer

if **myAnswer** is greater than the **secretValue** print

```
The SecretValue is smaller than ???
```

where ??? is the value of MyAnswer .

15. Embed the instructions written in 11. to 14. in a do{}while(); loop (CM_langage_C_1_2_conditions_et_boucles.pdf).

The condition will be

```
while(myAnswer   is different of the secretValue AND the maxRunNumber is
greater than runNumber).
```

16. After this do{}while(); loop, test if **myAnswer** is equal to the **secretValue** then print

```
Congratulation
```

else print

```
Bybye, Try again.
```

17. Remove the print of **secretValue**, see point 10.

18. Move the CLS instruction and copy it after the my_fflush(); instruction.

Exercice 1 – What is the day today

Exercice 1.1

1. Create a program asking for the year of birth. It will be registrated in a variable **birthYear** of type int. You must be sure that the value is belonging to [1900 ; 2020]. If a typo occurs the user is invited to enter a correct value.

2. Complete by asking for the month of birth, the value will be registrated in a variable **birthMonth** of type int. It must belong to [1 ; 12]. If a typo occurs the user is invited to enter a correct value.

3. Complete by asking for the day of birth, the value will be registrated in a variable **dayofBirth** of type int. If the month belongs to the set {1,3,5,7,8,10,12} the day must belong to [1 ; value] with value =31, if the month belongs to the set {4,6,9,11} the day must belong to [1 ; value] with value=30. For february it must be checked if the **birthYear** is bissextile that is if the following condition (!(y%4) && y%100 || !(y%400)) is true, then the upper bound value =29, else value =28, and the day must belong to [1-value] as well.

Exercice 1.2

1. Complete the previous program by counting the number of the day between the beginning of the year and the **dayofBirth**, obviously it will depends of the month and the year of birth. For example the Februray 4th is the 35th day of the year, the result is displayed on the screen and registrated in a variable **numberofDayInBirthyear** of type int.

2. Complete by printing the number of days spent alive during the **birthYear**, the result must be displayed on the screen and registrated in a variable **dayAliveinBirthYear** of type int.

Exercice 1.3

1. Complete the program by asking to enter the today date, that is the **yearToday**, **monthToday**, **dayToday** variables.

2. Count the number of the day since the begining of the year, registrate the result in a variable **numberOfToday** of type int.

Exercice 1.4

Count the number of days between the entered birthday and the entered date of today. The result will be registrated in a variable **daysofLife** of type int.

Exercice 2 – Drawing trees

2.1 Simple Tree

Give a program which draws the following figure if the user has entered 3 for the left Figure (and 5 for the right Figure), the value is registrated in a variable **height** of type int which must belong to [MINHEIGHT-MAXHEIGHT], where MINHEIGHT is a defined constant set to 2 and MAXHEIGHT is a defined constant set to 7.

Figure with **height**=3 Figure with **height**=5

2.2 Reverse Tree

Same constraints as above but this time to draw the following trees :

Figure with **height**=3 Figure with **height**=5

2.3 Triangle

Same constraints as above but this time to draw the following trees :

Figure with **height**=3 Figure with **height**=5

2.4 Adding a menu

Finish your program by adding the following menu:

```
h : set the height
s : drawing a simple tree
r : drawing a reverse tree
t : drawing a triangle
q : quit
```

If the user enter an unknown command, display an error message. Note that you can use the **pause()** instruction (CM_langage_C_0_toolbox.pdf) to wait for the user to press enter (to clear the screen and display the menu for instance).
When dealing with menu, most of the time it is better to use the **switch()** instruction (CM_langage_C_0_toolbox.pdf) than if(){}else if(){}....

Exercice 3 – grid manipulation

Exercice 3.1

Create a program asking for a number of rows registrated in a variable **nbRow** which must belong to [1 ; 20] and a number of column registrated in a variable **nbCol** which must belong to [1 ; 10]. The program must draw a rectangle with the index line and column, below an example is drawn with **nbRow**=3 and **nbCol**=2.

```
    1    2
1
2
3
```

Exercice 3.2

Modify the program such that the rectangle be filled with small letter obtained randomly and belonging to ['a' ; 'z']. Below an example :

```
    1    2
1   e    z
2   v    c
3   x    k
```

To obtain a random integer value between [LowerBound ; UpperBound] you should use the function rand(), which yields a random integer uniformly distributed, in the following manner :

```
value = lowerBound+ rand()%(1+upperBound-lowerBound);
```

e.g., if **lowerBound**=3 and **upperBound**=10, then **value**=3+rand()%8;

indeed rand()%8 yields a random value modulo 8, hence belonging to [0-7].

To transform the integer value in a char you can use the following instruction

```
char letter;
int value;
value=rand()%26;
letter='a'+value;
```

Exercice 4 – The doctor app’

Exercice 4.1

Write a program to help doctors in their diagnosis. The program requires the entry of the patient's weight (**weight** in kg belonging to [5 ; 200]) and height (**height** in m belonging to [0.4 ; 3]). The software calculates the body mass index (**imc=weight/(height*height)** in kg.m-2) and the category to which the patient belongs according to the categories in the table provided below.

IMC (kg m−2)	Interpretation
less than 16,5	undernutrition or anorexia
16,5 to 18,5	thinness
18,5 to 25	normal weight
25 to 30	overweight
30 to 35	moderate obesity
35 to 40	severe obesity
more than 40	morbid obesity

Exercice 4.2

Complete the software in order to compute some other indicators.

The Body Surface Area index (**bsa =(height*weight/3600)1/2**) and the Index of Body Surface Area (**ibsa=height^2 * wc/(vtc * bsa)**) for this purpose the Waist Circumference (**wc** in m belonging to [0.2 ; 2]) and the Vertical Trunk Circumference (**vtc** in m belonging to [0.4 ; 3]) must be entered.

These indicators allow the doctor to complete the diagnosis and help assess the dose of medication to be administered to the patient.

Exercice 4.3

Complete the software in order to create the following menu (switch case,):

```
1. Entry the weight and size of the patient
2. Display the IMC and the category of the patient
3. Entry the WC and the VTC
4. Display the BSA and the IBSA
5. Quit
```

The push ups app - examen

L'objectif est de développer une application d'entraînement pour les sportifs désirant travailler leurs pompes.

L'application doit proposer le menu suivant :

```
My push up training app
     s : set your maximal number of push ups
     d : display the training
     q : quit the application
```

L'utilisateur doit dans un premier temps saisir son niveau en indiquant le nombre maximal de pompes qu'il est capable de faire (menu s).

Ensuite il peut afficher le planning de son entraînement à l'aide du menu d.

Si l'utilisateur veut fermer l'application il doit choisir la commande q.

Il vous est demandé de compléter le programme donné en annexe pour répondre aux différentes contraintes du cahier des charges.

Travail à faire

1. Écrire la ou les instructions qui permettent l'affichage du menu présenté plus haut.

2. Écrire la ou les instructions qui permettent de récupérer la commande de l'utilisateur ands la variable **choice**.

3. Écrire les instructions qui permettent, si l'utilisateur choisit la commande 's':

- d'afficher le message suivant:

```
  What is your max number of push ups?
```

- de récupérer le niveau de l'utilisateur dans la variable **max_rep**.

4. Il faut tester le niveau saisi par l'utilisateur. Si l'utilisateur n'arrive pas à faire au moins 1 pompe, il faut indiquer le message suivant :

```
You need to be able to do at least one push up to use this app
```

et mettre la variable **too_weak** à 1.

Si l'utilisateur arrive à faire plus que **NBMAXREPMAX** pompes, il faut afficher le message suivant:

```
You are too strong, you do not need this app
```

et mettre la variable **too_strong** à 1.

Ecrire les instructions qui permettent de satisfaire ce comportement.

5.1. Pour pouvoir afficher le programme d'entraînement il faut d'abord que l'utilisateur ait saisi son niveau. Écrire la ou les instructions qui permettent de tester si la variable **max_rep** est strictement positive. Si ce n'est pas le cas, il faut afficher le message suivant:

```
You need to set your max rep first
```

5.2. Si l'utilisateur a bien saisi son niveau, il faut afficher son entraînement. L'affichage doit être de la forme :

```
here is your training for 10 days:
day 1 : 7, 4, 3
day 2 : 9, 6, 5
day 3 : 10, 7, 6
day 4 : 12, 8, 7
day 5 : 13, 9, 8
day 6 : 15, 11, 9
day 7 : 16, 12, 10
day 8 : 18, 13, 12
day 9 : 19, 14, 13
day 10 : 21, 16, 14
```

Le nombre de jours d'entraînements (10 dans l'exemple précédent) est défini par la constante **NBTRAININGDAY**. Pour chaque jour, on retrouve toujours 3 sets de pompes à réaliser :

- Le nombre de pompes du premier set est défini par l'équation : **(max_rep+i)/2+1+i**, i étant le jour du set
- Pour le deuxième set on a ; **(max_rep+i)/4+1+i**
- et pour le troisième set : **(max_rep+i)/6+1+i**

Par exemple, pour le jour 4, en supposant que l'utilisateur a saisi 10 pour **max_rep**, on a ;

- premier set = (10+4)/2+1+4 = 12
- deuxième set = (10+4)/4+1+4 = 8.5 => 8
- troisième set = (10+4)/6+1+4 = 7.33 => 7

Écrire les instructions qui permettent d'afficher un entraînement (faire une boucle sur le nombre de jours et pour chaque jour calculer les 3 sets).

6. Si l'utilisateur ne rentre aucune des trois commandes du menu, il faut afficher le message

```
X is not a valid command
```

avec X la commande rentrée par l'utilisateur. Écrire la ou les instructions qui permettent d'afficher ce message d'erreur.

7. L'application doit se fermer (sortir de la boucle) si l'utilisateur choisit la commande 'q', ou si l'utilisateur est trop faible (**too_weak** vaut 1) ou si l'utilisateur est trop fort (**too_strong** vaut 1). Proposer une boucle avec sa condition pour satisfaire cette contrainte.

Annexe

```
#include <stdio.h>
#if defined(WIN32) || defined(WIN64)
    #define CLS system("cls")
#else
    #define CLS printf("\033[2J\033[1;1H")
#endif

void my_fflush();

#define NBMAXREPMAX 15
#define NBTRAININGDAY 10

int main(void){
    char choice = 0;
    int max_rep = -1;
    int i;
    int too_weak = 0, too_strong = 0;

    // 7. boucle
        // 1. Afficher le menu

        // 2. Récupérer la commande de l'utilisateur

        CLS;

        if(choice == 's'){

            // 3. Récupérer le niveau de l'utilisateur

            // 4. Tester le niveau de l'utilisateur
        }
        else if(choice == 'd'){

            // 5.1. Tester si l'utilisateur a saisi son niveau

            // 5.2. Afficher l'entraînement
        }
        else if(choice != 'q'){
            //6. Indiquer que la commande n'est pas valide
        }
    // 7. boucle

    printf("Bye bye\n");
}

void my_fflush(){
    char c;
    do{ c = getchar(); }while(c!='\n');
}
```

Exercise 5 : An array introduction

(CM_langage_C_2_0_tableaux.pdf)

Exercise 5.1

1. Create an array (of **float**) named **tabScore** with a capacity of **NBMAX** floats (**NBMAX** being a constant set at 16).

2. Initialize the array with 5 random values between 0 and 20. The current number of scores (floats) in the array should be saved in a **nbScore** variable.

3. Display all the value of the array as follow (use a loop). Note that as the values are random, they could be not the same...

```
tabScore[0] = 12.0
tabScore[1] = 7.0
tabScore[2] = 13.0
tabScore[3] = 14.0
tabScore[4] = 2.0
```

Exercise 5.2

1. Compute and display the mean value (**meanvalue**) of the array (sum of all the scores divided by the number of scores).

2. Compute and display the variance (**variance**) of the array. The variance is the average of the (xi-meanvalue)^2 where meanvalue is the mean value of the data (previouslsy computed).

3. Compute and display the standard deviation (**std_dev**) of the array. The standard Deviation is the square root of the variance, a kind of mean value of the difference between the data and the mean value.

Exercise 5.3

Create the following menu:

```
1. Initialize randomly the array
2. Initialize the array score by score
3. Display the array
4.  Compute  and  display  the  mean  value,  the  variance  and  the  standard
deviation
5. Quit
```

1. Allows to randomly initialize the array. First you ask the user how many random values are needed (it must be between 1 and NBMAX, otherwise ask again). Then the array is filled with random values between 0 and 20.

2. Allows the user to give the scores one by one. First you ask the user how many scores the user want to enter (it must be between 1 and NBMAX, otherwise ask again). Then ask the user to user the values one by one (each value must be between 0 and 20, otherwise the user must enter a new one).

3. Display the array as presented in exercise 5.1.

4. Compute and display the mean value, the variance and the standard deviation. Note that if the array is empty, it should display the following error message:

```
You need to have a least one score to do that
```

5. Close the program.

5. If the user enters an unknown command, an error message should be displayed.

Exercise 6 : Array manipulation

Exercise 6.1

1. Create an array named **tab** with a capacity of **NBMAX** integers (**NBMAX** being a constant set at 20).

2. Initialize the array with 5 random numbers between 1 and 100.

3. Display the array as follow

```
tab = [50, 10, 7, 8, 12]
```

Exercice 6.2

The objective is to have the following behavior:

```
tab = [85, 38, 17, 25, 30]

Value to add at the end of the array:99
tab = [85, 38, 17, 25, 30, 99]

Value to add at the begining of the array:1
tab = [1, 85, 38, 17, 25, 30, 99]

Removing the last element of the array
tab = [1, 85, 38, 17, 25, 30]

Removing the first element of the array
tab = [85, 38, 17, 25, 30]

Inverting the value of the first and the last elements
tab = [30, 38, 17, 25, 85]

Sorting the array (selection sort)
tab = [17, 25, 30, 38, 85]
```

1. Ask the user to enter an integer and add it at the end of the array. Display the array to check if it worked.

2. Ast the user to enter an integer and add it at the beginning of the array. Display the array to check if it worked.

3. Remove the last value of the array. Display the array to check if it worked.

4. Remove the first value of the array. Display the array to check if it worked.

5. Invert the first value of the array and the last one.

6. Sort the array from using the selection sort algorithm (“The algorithm finds the minimum value, swaps it with the value in the first position, and repeats these steps for the remainder of the list.” - https://en.wikipedia.org/wiki/Selection_sort).

Exercice 7 – String introduction

1. Create a string **mystring** with a maximal capacity of **LGMAX** equals to 15

2. Ask the user to enter a sentence, save it into the **mystring** string.

3. Remove the '**\n**' at the end if it exists

4. Print the string

5. Change all the maj letter into min letters.

6. Print the string. For instance

```
Enter a sentence: Hello les PEIP2

before modification:
Hello les PEIP2

after modification:
hello les peip2
```

7. Format the sentence into an "American Title" style (all the first letters of words should be capital, not the others. For instance:

```
Enter a sentence: Hello les PEIP2

before modification:
Hello les PEIP2

after modification:
hello les peip2

after american title:
Hello Les Peip2
```

Exercice 8 – The MixedWord game

1. Create a string **hiddenword** with a maximal capacity of **LGMAX** equals to 20

2. Ask the user to enter a word, check if the entry is OK, otherwise ask again (OK means only composed of letters, no space, no number...)

3. Copy the word into a **mixedword** string (same capacity as the **hiddenword**. To copy a string you can use the **strncpy()** function (**string.h**)

```
char * strncpy ( char * destination, const char * source, size_t num );
```

- **destination**: the string where to copy the source
- **source**: the string to copy into destination
- **num**: the capacity of the destination

4. Randomly mix the letters of the **mixedword** srting. You can use **rand()** to generate random index values. Display the mixed string.

5. Ask the user to find the word corresponding to the mixedword. Use **fgets()** to get a string from the keyboard.

6. Compare if the given string corresponds to the hiddenword. You can use strcmp() to compare two strings:

```
int strcmp ( const char * str1, const char * str2 );
```

- str1: the first string to compare
- srt2: the second string to compare
- returns 0 if the two strings are equals (content)

7. Use a loop to ask the user to guess the work at most **NMAX** times. End the loop if the user fail **NMAX** times or if he founds the correct word.

8. Add a menu to the application:

```
The mixed word game
     1. Enter a word
     2. Find a word
```

Before allowing to find the word you need to check if the user already entered a word

Exercice 9 – The Hangman game

1. Create a string **hiddenword** with a maximal capacity of **LGMAX** equals to 20

2. Ask the user to enter a word, check if the entry is OK, otherwise ask again (OK means only composed of letters, no space, no number…)

3. Set all the letters of **hiddenword** into maj letters.

4. Create a string **foundword** with the same capacity and lenght as the **hiddenword** string but composed of '_'

5. Ask the user to enter a letter, and verify the entry (if not a letter ask again)

6. Check if the letter is in the **hiddenword**, if it is, display it in the **foundword** (instead of the '_').

7. Loop the user entry until he found the word or he made more than **NMAXTRIES** = 6. Display a result message saying if the player won or lose.

8. Add a menu to the application

```
The hangman game
     1. Enter a word
     2. Find a word
```

Before allowing to find the word you need to check if the user already entered a word

9. At the step 5, also check if the user already test a letter and display all the previously tested letters. To do that you can declare an int array name **usedletters** with a capacity of 26 and initialized with 0 values. Each time a letter is tested, put 1 into the corresponding value of the array (for instance if the user test the letter **'A',** do **usedletters[0] = 1;**)

10. Draw the hangman while the user test letters.

```
 o
\|/
 |
/ \
```

Example of display

```
  o
 \|
  |


tested letters so far:A E L O P S U
< POL__E__ >

Test a letter :
```

Remark: to be able to draw the "hangman", one way to process is to declare the drawing as a string a modify the displayed caracters according to the number of tries.

for instance, the following code gives the following display:

```
#include<stdio.h>

int main(){
     char drawing[13];
     int i;

     for(i=0; i<12; i++){
           if((i+1)%4==0){
                 drawing[i] = '\n';
           }else{
                 drawing[i] = '.';
           }
     }
     drawing[12] = '\0';

     printf("empty drawing:\n");
     printf("%s", drawing);

     drawing[5] = '*';

     printf("adding a star:\n");
     printf("%s", drawing);
}
```

```
empty drawing:
...
...
...
adding a star:
...
.*.
...
```

Exercice 10 : Some functions

Here are some training functions. For each of this function you should

- Write the prototype of the function;
- Write the implementation of the function;
- Test the function by calling it propertly in the main function.

1. Write a function named **EuclidianDistance** which computes the distance between two points in a 3D space. The coordinates of the points are respectively x1, y1, z1 and x2, y2, z2. The euclidian distance is defined as sqrt((x1-x2)^2+(y1-y2)^2+(z1-z2)^2)

2. Write a function named **Capital2Small** transforming a capital letter in a small letter. This function returns 1 if a transformation occurred and 0 otherwise.

3. Write a function named **Small2Capital** transforming a small letter in a capital letter. This function returns 1 if a transformation occurred and 0 otherwise.

4. Write a function named **IMCEvaluation** that computes the IMC (weight/size^2) according to a weight and a size. This function returns 0 if the IMC can not be computed (size = 0), 1 otherwise.

5. Write a function named **ModArgofComplex** that computes the modulus and the argument of a complex number. This function receives two float variables which are the real and imaginary part of the complex number. The function returns 1 if the argument is not defined and 0 otherwise.

We recal that the argument(https://en.wikipedia.org/wiki/Argument_(complex_analysis) can be computed as follows:

$$Arg(x+iy)=atan2(y,x)=\begin{cases}\arctan(\frac{y}{x}) & \text{if} \quad x>0\\ \arctan(\frac{y}{x})+\pi & \text{if} \quad x<0 \text{ and } y\geq 0\\ \arctan(\frac{y}{x})-\pi & \text{if} \quad x<0 \text{ and } y<0\\ \frac{\pi}{2} & \text{if} \quad x=0 \text{ and } y>0\\ \frac{-\pi}{2} & \text{if} \quad x=0 \text{ and } y<0\\ undefined & \text{if} \quad x=0 \text{ and } y=0\end{cases}$$

And the modulus:

$$Mod(x+iy)=\sqrt{(x^2+y^2)}$$

Exercice 11 : Function and arrays

1. Write a function **fillScores** that randomly initializes an array of floats. It needs the array, the number of elements and the capacity of the array. If the capacity is 0, it returns 1. Otherwise, the functions randomly initializes **n** elements, **n** beeing a value randomly chosen between 1 and the capacity of the array. If the array was correctly initialized, the function returns 0. Note that the number of elements should be updated.

2. Write a function **printArray** that displays an array of floats as follow:

```
[10.00, 7.30, 12.80, -8.90]
```

3. Write a function **modifScore** that adds a given value to all the elements of an array of floats. It needs the array and the value to add (float).

4. Write a function **addScore** that adds a given value at the end of an array of floats. It needs the array and the value to add (float). Returns 1 if the value has been added, 0 otherwise (if there is no space available).

5. Write a function **average** that computes the average of a given array. This function returns 1 if the average has been computed, 0 otherwise (if the array has no value).

6. Write a function **standardDeviation** that computes the standard deviation of a given array. Note that this function should use the **average** function defined previously. It returns 1 if everything went well, 0 otherwise.

7. Write a function **centerArray** that subtracts the meanvalue of a given array to each entry of the array. Note that this function should use the **average** function defined previously. It returns 1 if everything went well, 0 otherwise.

Exercice 12: Functions and strings

1. Write a function **mygets** that:

- gets a string from the keyboard
- removes the \n if it is a the end of the string
- emptys the keyboard buffer if needed
- returns the size of the string

This function should use fgets.

2.1. Write a function **isCapitalLetter** that returns 1 if a given character is a capital letter and 0 otherwise

2.2. Write a function **capital2small** that changes all the capital letters of a string into smal letters. Note that this function should use the **isCapitalLetter** function. This functions returns the number of changed characters.

3.1. Write a function **isSmallLetter** that returns 1 if a given character is a smal letter and 0 otherwise

3.2. Write a function **small2capital** that changes all the small letters of a string into capital letters. Note that this function should use the **isSmallLetter** function. This functions returns the number of changed characters.

4.1. Write a function **isVowel** that returns 1 if the given character is a vowel and 0 otherwise. Note that the vowel could be small or capital letter.

4.2. Write a function **countVowels** that counts and returns the number of vowels in a given string. This function should use the **isVowel** function.

5. Write a function **setUSTitleStyle** that sets a string into the US title stype (the first letters of words are capital letters and all the other ones are small letters). This function returns the number of changed letters. Note that this function could use the **isSmallLetter** and **isCapitalLetter** functions.

6. Write a function **checkAndCountBracket** that checks if the number of opening and closing brackets are equals. If returns an integer corresponding to opening – closing brackets.

7. Write a function **commentString** that adds "/*" at the beginning of the string and "*/" at the end. It returns 0 if everything went well, 1 otherwise (if the string is not big enough for instance).

Exercice 13: Veni vidi vici

1. Write a function **getCase()** that tests the case sensitivity of a given character. This function returns 1 if the character is an uppercase, 2 if the character is a lowercase and 0 if the character is not a letter.

2. Write a function **shiftLetter()** that shifts a letter from a given index. For instance if the letter is 'C' and the shifting index is 3, the letter becomes 'F' (the 3rd letter after C). Note that this function should work with uppercase and lowercase. Here are some expected results:

- 'A' + 5 => 'F'
- 'w' + 5 => 'b'
- 'T' + 260 => 'T'
- 'D' - 2 => 'B' (the index can be negative)
- '%' + 3 => '%' (if the character is not a letter it should not be modified)

This function returns 1 if everything went well (the letter has been shifted), 0 otherwise (if the character is not a letter).

3. Write a function **cesarCoding()** that uses a Cesar code[1] to code a string message. This function needs the string to code and a shifting index. For instance, lets consider the string "Hello" and a shifting index equals to 3, the codded string become "Khoor".

4. Write a function **cesarDecoding()** that decodes a string coded with a Cesar Code. This funcion needs the string to decode and the shifting index. Note that the index the one used to code the message.

5. Write a function **letterHistogram()** that creates a histogram of the letters of a string. The histogram is an array of 26 integers, each entry containing the number of corresponding letter in the string. For instance **histogram[0]** will contain the number of **'A'** and **'a', histogram[5]** the number of **'f'** and **'F'** etc... According to the message 'Hello':

histogram[7] = 1 (H), histogram[4]=1 (E), histogram[11]=2 (L) and histogram[14]=1 (O)

6. Write a function **printHistogram()** that display a histogram. For each letter, if the number of occurences is not 0 it is displayed with '*' characters. For instance, the histogram computed with the string 'Hello' should be displayed as:

```
E/e: *
H/h: *
L/l: **
O/o: *
```

7. Write a function **mostFrequentValue()** providing the greatest value in the histogram and the index of this greatest value.

---

1 https://fr.wikipedia.org/wiki/Chiffrement_par_d%C3%A9calage

8. It is known that the **'e'** is the most used letter in latin language, the second frequently used is the **'a'** (https://fr.wikipedia.org/wiki/Fréquence_d'apparition_des_lettres_en_français). Write a function **hackCesar()**, that search for the most frequent character in a coded string. Then it assumes that this most frequent character corresponds to the 'e' and thus returns the index used to code the string.

Exercice 14: Che Guevara

You can find the code algorithm here: http://www.cryptage.org/chiffre-che-guevara.html

0. Write a function **getCase()** that tests the case sensitivity of a given character. This function returns 1 if the character is an uppercase, 2 if the character is a lowercase and 0 if the character is not a letter.

1. Write a function **printArray()** that display the content of an array of integers as

```
[2, 34, 54, -5, 12]
```

2. Write a function **valueIndex()** that returns the index in a array of a given value. If the value is not in the array, the function returns -1.

For instance, for the array [0, 2, 3, -5, 12, 8]

- If the value is 2 the function should return 1
- If the value is -5 the function should return 3
- If the value is 9 the function should return -1

3. Write a function **generateSubstitutions()** that computes the array of the letter substitution. For each of the 26 values (one for each letter) the function should set a value between 10 and 99. Each value should be only once in the array (you may use the **valueIndex** function).

Here is a substitution array that is correct:

```
[64, 18, 26, 21, 12, 41, 22, 77, 53, 19, 45, 17, 37, 62, 44, 86, 52, 95, 11,
28, 96, 67, 73, 32, 60, 46]
```

4. Write a function **generateKey()** that randomly generate a key (an array of integer and each value of the array should be in [0;9]. This function needs the array and the number of digit of the key. For instance, for a key with 5 values, the following array can be generated:

```
[ 1,  9,  0,  1,  6]
```

5. Write a function **insertSpace()** that insert a space in a string at a given index. For instance, considering the string ''coucou'', if we insert a space at the index 1, the string becomes ''c oucou''.

This function returns 0 if everything went well, 1 if the size of the string is not enought to insert the space, 2 of the index is not in the string.

This function should first compute the size of the string, then check if there is enought space, then check if the index is in the string, then shift all the char from the last one to the index, and finally insert the space at the given index.

6. Write a function **isNumber()** that returns 1 if a given character is a number, 0 otherwise.

7. Write a function **preCode()** that converts all the letter of a message to numbers regarding a substitutions array. The message (a string) should not contains any numbers. The function returns 1 if the message can not be precoded (it contains a number), 2 if the string capacity is not enought for the precoded message and 0 if everything went well.

This function should loop over the string. For each char, it should rest if the char is a number (**isNumber**) and return 1 if it is. If the char is not a number, it should test if it is a small letter (**getCase**) and convert it into a capital letter.

For each capital letter, the idea is to insert a space just after the letter (**insertSpace**), then get the ten of the corresponding substitution and replace the letter with it, and put the unity value into the inserted space.

This should give something like:

```
Subtitutions: [83, 26, 37, 35, 33, 56, 22, 79, 11, 42, 77, 60, 89, 86, 70,
46, 62, 81, 18, 97, 72, 90, 23, 47, 65, 69]
Key: [ 2,  2,  8,  9,  7]
Enter the message to code: this is well done
after preCode: "97791118 1118 23336060 35708633"
```

`'t' -> "97", 'h' -> "79", 'i' -> "11"...`

8. Write a function **generateStringKey()** that computes a key string regarding the message and the key array. The generated key string should have the same size as the message and each value of the key string corresponds to the values of the key array that are repeated. For instance:

```
Subtitutions: [83, 26, 37, 35, 33, 56, 22, 79, 11, 42, 77, 60, 89, 86, 70,
46, 62, 81, 18, 97, 72, 90, 23, 47, 65, 69]
Key: [ 2,  2,  8,  9,  7]
Enter the message to code: this is well done
after preCode: "97791118 1118 23336060 35708633"
keyString: "22897228 9722 89722897 22897228"
```

He the key is [2,2,8,9,7], then the Key String it the repetition of those numbers for each number in the precoded message.

9. Write the function cheCoding() that codes a message according to the string key. For each number of the message, the coded value is the number plus the correspond key in the keystring modulo 10: new value = (old value + key) % 10

This should provide the following result:

```
Subtitutions: [83, 26, 37, 35, 33, 56, 22, 79, 11, 42, 77, 60, 89, 86, 70,
46, 62, 81, 18, 97, 72, 90, 23, 47, 65, 69]
Key: [ 2,  2,  8,  9,  7]
```

```
Enter the message to code: this is well done
after preCode: "97791118 1118 23336060 35708633"
keyString: "22897228 9722 89722897 22897228"
Coded message: "19588336 0830 02058857 57595851"
```

Finally you can use the implemented cheDeCoding() function to test your coding function (you should have the same message). For instance:

```
Subtitutions: [79, 51, 14, 64, 10, 59, 62, 18, 65, 43, 76, 81, 50, 52, 29,
46, 53, 23, 36, 27, 19, 42, 63, 21, 84, 86]
Key: [ 7,  5,  7,  3,  1]
Enter the message to code: well done!
after preCode: "63108181 64295210!"
keyString: "75731757 31757317!"
Coded message: "38839838 95942527!"
deCoded message: "WELL DONE!"
```

Exercice 15: Grid and colors

For this exercice you have a base code that you should start with. This code allows to handle the colors with Windows and GNU/Linux based systems.

1/ **get_int_in_range** : Function that gets an integer from the keyboard. If the entry should be between two bounds (in a intevall). If the value given by the user is not between the bounds, the function asks the user to provide a new value. This is done until the user enter a correct value (a value that is in the given interval).

2/ **init_grid_with_random_letters** : This function allows to intialize a grid with random capital letters. A grid is an array (of char here) that is considered as a 2 dimensional array. For instance the grid:

```
[A, B, C, D]
[E, F, G, H]
[I, J, K, L]
```

corresponds to the array: [A, B,C, D, E, F, G, H, I, J, K, L] with 4 columns and 3 lines.

The functions **init_grid_with_random_letters** needs the array, the number of lines and the number of columns.

To evaluate an array as a 2D grid, you need to be able to compute the index of a value in the array regarding the row and column indexes. For instance, considering the following grid

```
   0  1  2  3
0 [A, B, C, D]
1 [E, F, G, H]
2 [I, J, K, L]
```

And the corresponding array [A, B, C, D, E, F, G, H, I, J ,K], the index of 'A' (row 0, column 0) is 0, the index of 'C' (row 0 column 2) is 2, the index of 'I' (row 2, column 0) is 8, the index of 'G' (row 1, column 2) is 6...

What is the relation between the index and the row and column?

3/ **print_grid** : This functions allows to display a grid as follow (for 6 columns and 5 rows):

```
  012345
0|DGFALS
1|HZZTZN
2|LQTGTV
3|LTKVZZ
4|ORDZFS
```

3/ **print_grid_color** : This functions allows to display a grid as follow (for 6 columns and 4 rows):

```
   0  1  2  3  4  5
0| T  M  S  M  T  L
1|  D  J  C  L  N  B
2| E  U  I  X  F  G
3| O  G  F  Y  D  B
```

The colors must be generated randomly with the function get_random_color.

Note 1: you should not have the same color for the text and the background...

Note 2: There are spaces between the letters

Recursivity

A recursive function is a function that calls itself. For instance:

Exercice 16: Some basics

For those functions you are not allowed to use loops (for, while, dowhile).

1. Write a function **my_pow()** that computes the power of a float number

2. Write a function **my_factorial()** that computes the factoriel of a float number

3. Write a function **sumArray()** that returns the sum of all the integers of an array.

3. Write a function **printArray()** that prints an array of integer (starting by the last value).

Exercice 17: Quick sort

1. Write a function **swap()** that swaps the values of two integers.

2. Write a function **partition()** that receives an array of integers and a pivot. The function organises the array such that all the numbers lower that the pivot are at the beginning of the array and all the numbers greaters that the pivot are after the pivot. This function returns the new index of the pivot.

3. Write a recursive function **quick_sort()** that uses the **partition()** function to sort an array of integers.