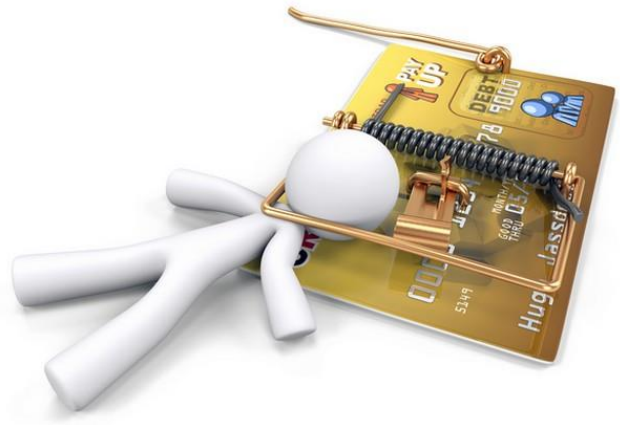

Pour aller plus loin avec les microcontrôleurs tels que, Arduino et AVR connexes, il est indispensable de posséder de bonnes bases du langage C++.

Ainsi, une fois n'est pas coutume, je vous propose le petit programme ci-joint, qui est la traduction en C++ d'une curiosité mathématique découverte lors d'un voyage dans le labyrinthe de la toile …

L'algorithme de Kaprekar

En préambule, quelques mots sur ce monsieur Kaprekar.

Dattatreya Ramachandra Kaprekar est un mathématicien indien connu pour ses recherches sur la notion de nombre de Kaprekar ainsi que l'algorithme de Kaprekar. Boudé par ses contemporains, ses travaux seraient passés inaperçus s'ils n'avaient pas été relayés par Martin Gardner, spécialiste de mathématiques récréatives.

L'algorithme de Kaprekar est un processus qui transforme un nombre entier en un autre nombre entier.

Il fonctionne de la façon suivante : soit N (ici 4 chiffres) un nombre entier. Soit Nd le nombre obtenu en rangeant les chiffres de N dans l'ordre décroissant, et Nc le nombre obtenu en les rangeant dans l'ordre croissant. L'Algorithme de Kaprekar retourne alors le nombre Nd - Nc. On réitère le processus plusieurs fois jusqu'à obtenir toujours le même nombre...Soit **6174**

Combien d'itérations sont-elles nécessaires, si on part d'un nombre à quatre chiffres ? Quels sont les résultats intermédiaires ?

C'est ce que je vous propose avec ce petit programme, très documenté, écrit en C++ d'après un canevas trouvé sur le net.

Attention, tous les nombres divisibles par 1111 ne fonctionne pas, évidemment.

J'apprécierais beaucoup si quelqu'un pouvait coder cet algorithme en Python ou en un autre langage.

```
/* Programme C++ pour trouver le nombre d'itérations de la
routine pour atteindre 6174 (constante de Kaprekar).
Ce programme retourne une erreur pour les entrées non valides
(par exemple les nombres divisibles par 1111*/

#include <bits/stdc++.h>
using namespace std;

int cnt = 0; //compteur d'itérations
// Test de la validité du nombre entré
bool estValide(string& nombre, int& n)
{

// Stocke chaque chiffre dans un ensemble
unordered_set<char> freq;
for (int i = 0; i < n; i++)
freq.insert(nombre[i]);
// Return false si tous les chiffres sont les mêmes sinon true
return freq.size() >= 2 ? 1 : 0;
}
int Kaprekar_Cste(string nombre, int n)
{

// Lorsque le longeur du nombre est supérieure à 4, ou que le
nombre possède 4 chiffres, mais identiques
if ((!estValide(nombre, n) || n > 4) && cnt == 0)
{
cout<<"Le nombre choisi est invalide"<<endl;
return -1;
}
else
{
if (stoi(nombre) == 6174) //convertit le string en un entier
{
return cnt; //retourne le nombre d'itérations
}
}

// Compte le nombre d'itérations
cnt++;

// Si le nombre entré possède moins de 4 caractères, on insère
un 0 tout à gauche

while (n++ < 4)
nombre.insert(0, "0");
string nombre2 = nombre;

// Fabrication du plus petit nombre (ascendant 1 2 3 4 5) et
du plus grand nombre (descendant 5 4 3 2 1)
sort(nombre.begin(), nombre.end()); //ordre ascendant
```

```
sort(nombre2.begin(), nombre2.end(), greater<int>()); //ordre
descendant

// Conversion string en integer
int increasing = stoi(nombre); //difficile de trouver un terme
simple en français
int decreasing = stoi(nombre2);
// Soustraction du plus grand nombre moins le plus petit
string res_soust = to_string(abs(increasing - decreasing));
cout<<"ressoust = "<<res_soust<<endl;
// Si la valeur 6174 n'est pas atteinte on réitère le
processus, sinon on stoppe le processus
return Kaprekar_Cste(res_soust, res_soust.length());
}

// Nombre à traiter
int main()
{
string nombre;
cout<<"Choisissez un nombre de 4 chiffres"<<endl;
cin>>nombre;
cout<<"le nombre choisi est "<<nombre<<endl;
int n = nombre.length();
// Appel de la fonction Kaprekar_Cste
Kaprekar_Cste(nombre, n);
cout<<"le nombre d'iterations necessaires pour atteindre 6174
est de "<<cnt<<endl;
return 0;
}
```

```
Choisissez un nombre de 4 chiffres
8631
le nombre choisi est 8631
ressoust = 7263
ressoust = 5265
ressoust = 3996
ressoust = 6264
ressoust = 4176
ressoust = 6174
le nombre d'iterations necessaires pour atteindre 6174 est de 6

Process returned 0 (0x0)   execution time : 3.344 s
Press any key to continue.
```