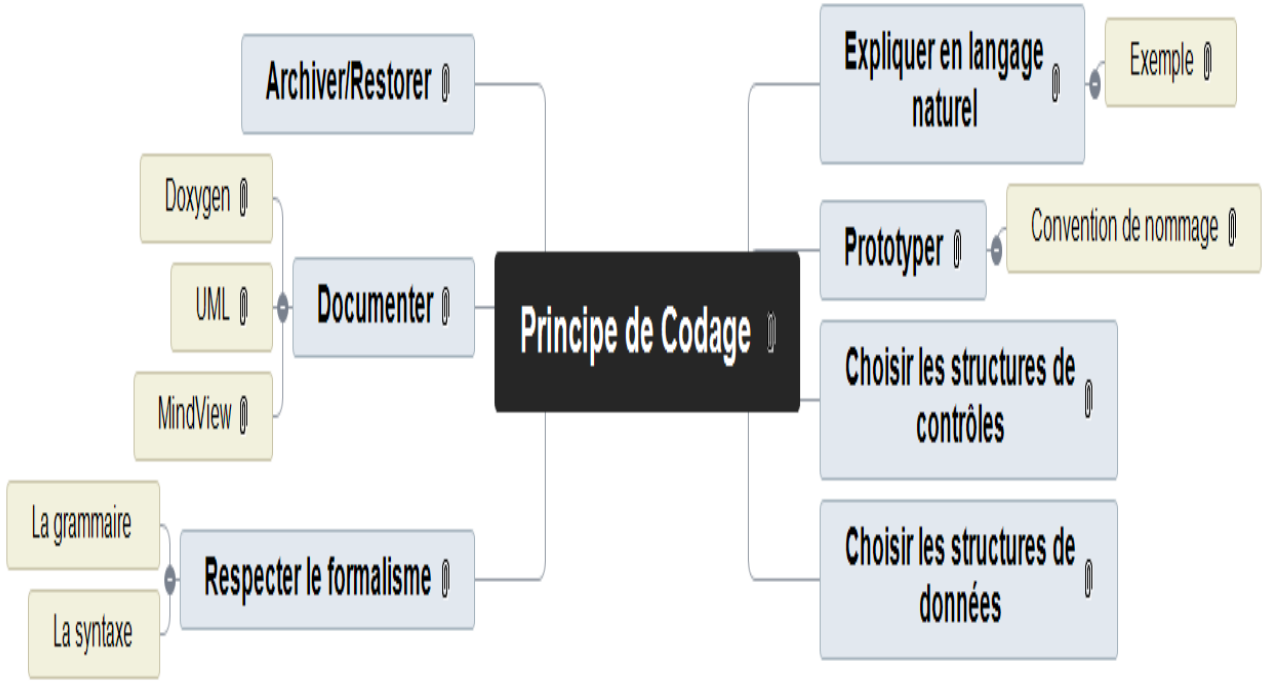
PRINCIPE DE CODAGE



Coder dans un langage donné, ici en C ou C++, revient a expliquer dans le langage concerné sa compréhension d'une résolution d'une situation particulière.
Il ne faut donc pas coder pour coder mais bien expliquer en français sa perception de la situation à résoudre en partant du général pour aller au particulier.

1 EXPLIQUER EN LANGAGE NATUREL

Que cherchez vous à résoudre comme situation.?
Il faut bien comprendre ce que vous voulez obtenir au final. En précisant au maximum les détails. Évitez les formulations trop vague (ex: "je veux coder mon projet")

Comment allez vous vous y prendre ?
Pour la situation a laquelle vous êtes confronté vous devez expliquer pas à pas, étapes élémentaires les unes après les autres du comment vous "voyez", "percevez", "resolvez" la situation, le problème qui vous apparaît.

Est ce que vos explications (en Français) répondent bien à l'objectif recherché ?
Vous êtes la seule personne a savoir si le problème initial que vous avez mis en évidence est résolu. Si vous ne pouvez même pas expliquer pourquoi selon vous le problème est correctement résolu, preuve à l'appui, comment voulez vous que quelqu'un d'autre y arrive ?

EXEMPLE

J'ai un fichier qui contient des lignes de chiffres. Je ne sais pas combien de lignes. Mais je sais que chaque ligne a 13 chiffres. Chacun de ces chiffres pouvant être entre 1 et 125.

Je cherche a savoir combien de découpage d'apparition des 125 chiffres je peux faire et dans chaque découpage combien de fois chacun des chiffres apparaît.

Ex de fichier
1,5,4,8,7,121,2,99,45,85,77,69,48
23,48,2,1,77,57,44,3,9,10,11,25,6

Dans ce cas particulier je ne peux faire qu'1 découpage (je n'ai pas pu "voir la totalité des 125 chiffres car pas assez de données)
Mais je jeux dire :<chiffre>:<nombre de fois vu>,

(1:2),(5:1),(4:1),(8:1),(7:1),(121:1),(2:2),(99:1),(45:1),(85:1),(77:2),(69:1),(48:2),(23:1),(57:1),(44:1),(3:1),(9:1),(10:1),(11:1),(6:1)

Comment mon cerveau a t'il fait pour trouver cette solution. Je dois analyser pas a pas.

J'ai :
- trouvé le fichier (ex de fichier)
- lu le fichier une ligne après l'autre
- pour chaque ligne :
    - j'ai regardé le chiffre
    - vérifié qu'il était bien entre 1 et 125
    - compté combien de fois j'avais déjà vu le chiffre en cours
    - vérifié que la ligne avait bien uniquement 13 chiffres
    - puis je suis passé à la ligne suivante

Dans cet exemple "automatiquement" je me suis dit je n'ai pas "vu" tous les 125 chiffres donc je n'ai pu faire qu'1 découpage (partiel).

Si mon fichier était plus grand, il aurait donc fallu que je compte si j'ai "vu" tous les 125 chiffres avant de commencer à compter pour le découpage suivant.

Maintenant que j'ai fait un cas particulier je peux essayer une généralisation.

```
///     Récupérer toutes les lignes
///     Se positionner sur la première ligne
///     Tant qu'il y a des lignes à lire
///     Regarder chaque chiffre
///     Est ce le début d'une détermination de découpage ?
///             Oui alors ?
///                 Se rappeler le numéro de la ligne commençant le découpage
///             Non alors ?
///                 Ne rien faire
///     Vérifier si le chiffre est bien entre 1 et 125
///     Ai  je déjà vu tous les 125 chiffres ?
///     Non alors ?
///             Ai je déjà vu ce chiffre ?
///             Non alors ?
///                                 Dire que maintenant ce chiffre est vu
///                 Que c'est la première fois qu'on le voit
///                 Que c'est un des chiffres compris entre 1 et 125
///             Oui alors ?
///                 je vois donc ce chiffre une fois de plus
///     Oui alors ?
///             Se rappeler le numéro de la ligne terminant le découpage
///             Quel est la position sur la ligne qui a permis de dire un découpage est terminé ?
///             Cette position est elle inférieure au maximum des positions possible sur la ligne ? (13)
///             Non alors ?
///                         On était donc sur le dernier élément de la ligne, le prochain découpage va commencer à la ligne suivante (position 1)
///             Oui alors ?
///                 Le prochain découpage commence sur cette même ligne, mais a la position suivante
///     Ai je traité toutes les lignes du fichier ?
///     Non alors ?
///                     Continuer l'analyse
///     Oui alors ?
///             J'ai résolu mon problème.
///     Fin
```

Comme vous pouvez le voir, je pars du général (le fichier), pour arriver au particulier (la ligne).
J'explique en Français ma solution

2 PROTOTYPER

En se basant sur l'exemple et selon le degrés de liberté que vous voulez mettre dans la résolution du problème il y aura plus ou moins de choses fixes et/ou variables.
Vous définissez ainsi **l'API** (***A**pplication **P**rogram **I**nterface*) de votre réponse au problème initial.

Au plus simple on veut vouloir dire combien de découpage il est possible d'avoir dans le fichier.
Ainsi formulé un prototype possible serait:
***int* trouverNombreDecoupageDansFichier(char * ch_nomDuFichier) ;**

En donnant d'autres libertés (Ramener la liste des découpages, d'un fichier ayant des lignes de i_nbChiffre, compris entre i_minDuChiffre et i_maxDuChiffre), un prototype possible :
Liste* trouverListeDesDecoupages(char *ch_nomDuFichier, int i_nbChiffre, int i_minDuChiffre , int i_maxDuChiffre) ; *Liste étant un nouveau type que vous auriez à définir.

C'est votre liberté de programmeur. En cohérence avec avec l'objectif de traitement et les conventions d'écritures des fonctions.

2.1 CONVENTION DE NOMMAGE

Pensez à retirer les accents et c cédille dans tous vos commentaires...

Pour chaque projet les conventions à appliquer sont les suivantes :

Nom de Fonctions :

e<n° Élève >< numéro projet (1er lettre Majuscule)>< Abréviation projet 1er lettre Majuscule>_<verbe en minuscule>< complément avec chaque 1er lettre des mots en majuscule>(....);
(utiliser QT ou Éclipse pour renommer efficacement vos nom de fonctions)

N° Projet	Nom du Projet	Abréviation Projet
CSG1	Marche Arrêt Circuit Climatiseurs	**MACC**
CSG2	Pager Mobile de Taches	**PMT**
CSG3	Douchette Portable	**DP**

Exemple :
int e1Csg1Macc_trouverNombreDecoupageDansFichier(char * ch_nomDuFichier);
int e1Csg2Pmt_trouverNombreDecoupageDansFichier(char * ch_nomDuFichier);
int e1Csg3Dp_trouverNombreDecoupageDansFichier(char * ch_nomDuFichier);

Type :
Quand vous créez un nouveau type :
E<n° Élève >< Abréviation type de base en minuscule>< Nom avec chaque 1er lettre des mots en majuscule>(....);

Ex:
E1cMaClasse;
E1stMaStructure;
E1enMonEnum;
E1qsMonQstring;
E1qtvMonQtableView;

..
Ainsi vous saurez que dés que vous voyez un **"e"** c'est une fonction, et un **"E"** c'est un type.

Variables:
Pour vos variables internes une plus grande liberté est autorisée. Mais si possible respecter minuscule puis 1ere lettre des mots en majuscule ex:

```
bool  b_isOk = false;
int i_maVariable_1;
Qstring qs_maVariable_1;
```

Le principe est de mettre le **type de la variable abrégé en minuscule, un souligné,** puis **un nom significatif** pour le rôle de la variable.
A harmoniser pour le groupe de projet, voir la classe.

Pour les variables courtes temporaires n'ayant pas une porté significative pour la compréhension du code, il n'est pas recommandé de mettre le type de la variable dans la déclaration.

Ex: on mettra **int i;**(et non pas int i_i;)

3 CHOISIR LES STRUCTURES DE CONTROLES

Une fois avoir exprimé la résolution en francais, l'avoir faite lire par quelqu'un d'autre, l'avoir amendé et validé, il ne reste plus qu'a l'exprimer dans le langage de programmation pour assurer les articulation de votre pensée.
If/else;while;do/while;for;..

4 CHOISIR LES STRUCTURES DE DONNEES

Ici c'est le comment lier votre pensée :
Listes chaînées; tableaux, variables, structures, classes, unions,...

5 RESPECTER LE FORMALISME

Chaque langage de programmation à 2 composantes à respecter scrupuleusement afin d'etre compris. La syntaxe (les mots clefs sont correctement ecrit) et la grammaire (les mots sont correctement agencés dans la "phrase").

6 DOCUMENTER

En dehors de la présentation de soutenance réalisée sous PowerPoint ou équivalent,

Il doit être normalement livré :

- un document **programmeur** (pour que votre travail puisse être maintenu, compris, continué...)
- un document **utilisateur** (les procédures à exécuter pour atteindre les fonctionnalitées du projet)
- un document **installateur** (éventuellement dans le cas ou il n'y aurait pas simplement un seul executable)

Pour vous aider à la rédaction de ces documents vous devez utiliser les ressources suivantes mise à disposition.

Ce ne sont la que des outils.

En aucun cas être un expert de ces outils ne remplacera la production finale du projet.

Mais ils peuvent vous aider considérablement.

6.1 DOXYGEN

Cet outil, si vous avez documenté votre code avec les balises adéquate vous permettra de générer la documentation en PDF ou l'HTML La navigation dans la logique et les fichiers de votre réalisation en sera grandement facilité.

Liens utiles :
http://www.stack.nl/~dimitri/doxygen/download.html
https://graphviz.gitlab.io/download/
http://www.mcternan.me.uk/mscgen/
https://sourceforge.net/projects/msc-generator/

6.2 UML

Il faut absolument montrer en tant que programmeur montrer que vous savez maîtriser ce langage graphique et les allés retours vers et depuis le code généré ou écrit.

6.3 MINDVIEW

L'efficacité n'est plus a démontrer.
Cette présente note en est un exemple.

7 ARCHIVER/RESTORER

L'outil de suivit de version **GIT** est celui préconisé.
Il en existe d'autres. On doit pouvoir voir et revenir à tout moment à n'importe lequel des points de votre phase de développement.

Un exemple de sortie avec doxywizard :

My Project v0.1

synopsys