IUT Robert Schuman
Institut universitaire de technologie
Université de Strasbourg

M32 - Arithmétique et Cryptographie

Projet de cryptographie

Objectifs : Définir les normes de codage demandées dans le cadre du projet M32

Avant tout, allez lire ce très bel ouvrage ! ModernC, par Jens Gustedt

Règles à respecter pour le développement du projet

Makefile et compilation séparée

Un fichier `makefile` doit permettre une compilation séparée du projet à partir de la commande `make`.

Le projet doit être décomposés en plusieurs fichiers comme indiqué ci-dessous :

- `main.c` : fichier contenant uniquement la fonction `main()` qui ne devrait pas excéder une vingtaine de lignes
- `utils.c` : fichier contenant l'ensemble des fonctions "utilitaires" qui pourraient être utilisées par d'autres programmes
- `rsa.c` (par exemple) : fichier contenant les fonctions spécifiques à votre algorithme
- un fichier d'en-tête .h pour chacun

Nommage des variables et fonctions

Les variables et noms de fonctions sont **exclusivement en anglais** avec un sens exprimant clairement le rôle de la variable/fonction :

- en minuscules avec _ comme séparateur si plusieurs mots
- en majuscules avec _ comme séparateur pour les constantes

Exemples :

MAL !

```
int toto, wtf, *p, _Test;
char UnCaractere, une_ChaineDe-Caracteres[500];
```

BIEN !

```
float mean_square_error;
uint8_t index;
static const double PI=3.1415926535;
```

Attention : on préférera pour ce projet les types `int` standards (`uint8_t`, `uint16_t`, etc.) se trouvant dans la bibliothèque `stdint` (`#include <stdint.h>`).

Les fonctions

Chacune est décrite par un en-tête de commentaires :

- décrivant TOUS les paramètres de la fonction,
- et décrivant TOUTES les valeurs possibles renvoyées.

Toujours prévoir (quand cela a un sens) un code de retour spécifique pour les erreurs éventuelles.

Exemple de déclaration de fonction :

```
/*
 * Fonction effectuant la division euclidienne de a par b
 *  - a : dividende
 *  - b : diviseur
 *  - q : quotient de la division (paramètre de sortie)
 *  - r : reste de la division (paramètre de sortie)
 * Code de retour : 0 en cas de succès, -1 en cas d'erreur
 */
int euclidian_division(uint32_t a, uint32_t b, uint32_t* q, uint32_t* r) {
      ...
}
```

Tous les appels de fonction doivent être testés avec utilisation de la variable `errno` et de la fonction `perror`.

MAL!

```
fd=open(filename, O_RDONLY);
```

BIEN!

```
if ((fd=open(filename, O_RDONLY)) < 0) {
  perror("open");
  return(EXIT_ERROR);
}
```

Les variables "importantes" utilisées dans les fonctions sont déclarées au début de la fonction et commentées. Les variables plus locales le sont au fil de l'eau et commentées si nécessaire (inutile de commenter un index de boucle par exemple).

Une fonction ne devrait jamais excéder une trentaine de lignes.

Les entrées-sorties

Toutes les entrées-sorties (lecture et écriture de fichiers) se font **impérativement** avec les primitives `open`, `read`, `write` et `close`.

L'utilisation de la structure `FILE` est **interdite**!

Divers

- Toujours penser à fermer ce qui a été ouvert! `open` –> `close`.
- Toujours désallouer la mémoire allouée par `malloc`.
- Toujours utiliser `size_t` pour les index de boucle.
- Le code doit être parfaitement indenté (avec des tabulations et non des espaces).
- Tout menu ou interaction avec l'utilisateur est **TOTALEMENT interdit et inutile!**