TD #1 — Flot de contrôle, tests, analyse de coût (tout ça ?!)

Exercice 1. *Affichage joli*

On se donne le programme C suivant :

```c
void pretty_print(uint32_t x) {
    if (x > 1000000000) {
        printf("%u ", x / 1000000000);
        x = x % 1000000000;
    }
    if (x > 1000000) {
        printf("%u ", x / 1000000);
        x = x % 1000000;
    }
    if (x > 1000) {
        printf("%u ", x / 1000);
        x = x % 1000;
    }
    printf("%u\n", x);
}

int main(void) {
    uint32_t a = 1;
    int i = 0;
    while (i < 9) {
        a = 2 * a;
        i = i + 1;
    }
    pretty_print(a);

    return EXIT_SUCCESS;
}
```

1. Quels fichiers d'en-tête faudrait il lui ajouter pour qu'il compile ?
2. Exécutez ce programme « à la main », et déterminez en chaque point d'exécution les valeurs des différentes variables *qui y sont visibles*, ainsi que ce qu'il affiche éventuellement sur la sortie standard.
3. Quelle autre construction aurait on pu utiliser pour implémenter la boucle `while` de la fonction `main` ?
4. Réécrivez cette boucle de cette façon.

5. Comment pourrait on modifier cette boucle afin d'exécuter *toutes* les lignes de la fonction *pretty_print* ?

Le spécifieur de conversion `%u` n'affiche pas les chiffres nuls, ce qui fait que la fonction *pretty_print* proposée n'est en fait pas correcte (pour son usage prévu). Il existe cependant une variante « `%03u` » qui permet d'afficher toujours au moins trois chiffres en insérant si besoin des zéros « à gauche ».

6. Proposez une version corrigée de *pretty_print*. Prenez garde à bien traiter tous les cas.
7. Le test effectué précédemment est-il toujours exhaustif ? Si non, proposez une fonction de test qui permette d'exécuter toutes les lignes de la nouvelle fonction *pretty_print*.

Exercice 2. *Kworème*

On se donne la fonction C suivante :

```c
void quorem(uint32_t x, uint32_t d, uint32_t out[2]) {
    uint32_t q = 0;
    uint32_t r = x;

    while (r >= d) {
        r = r - d;
        q = q + 1;
    }

    out[0] = q;
    out[1] = r;

    return;
}
```

1. Que semble-t'elle calculer ?
2. Ajoutez-y des assertions « `assert` » afin de vérifier à l'exécution que ses entrées sont (a priori) valides, et que le résultat produit est celui (a priori) attendu.
3. Cette fonction vous semble-t elle efficace ?
4. Avez-vous une idée de modification la rendant (beaucoup) plus efficace ?
5. Supposez que l'on arrive à prouver que la version ci-dessus est correcte (pour les spécifications imaginées). Proposez une démarche permettant de tester la correction de votre variante plus efficace.