TP1

1. Documentation de référence et compilation

a. *Documentation :*

Les docs en ligne :
http://www.infres.enst.fr/~charon/CFacile/comment/index.html
http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1570.pdf
https://fr.wikiversity.org/wiki/Langage_C

→ Plus complet, index de l'aide en ligne de la commande UNIX man
http://c.developpez.com/

b. *Outils et compilateur :*

→ Pour toutes les autres questions…
Tous vos programmes sont au départ écrits sous forme de texte et sauvegardés dans des fichiers de code source (extensions .c), créés ou modifiés avec un éditeur de texte. Vous utiliserez dans un premier temps gedit : http://projects.gnome.org/gedit/
La compilation est une succession d'étapes visant à transformer ce texte en un programme, c'est à dire un fichier de code exécutable par votre machine (une suite d'instructions de code binaire pouvant être chargée dans la pile d'instructions du processeur).

Un EDI (Environnement de Développement Intégré) est un logiciel consacré au développement d'applications. Un EDI vous permet d'éditer votre code source tout en bénéficiant de la coloration syntaxique, de le compiler, de l'exécuter et de le déboguer (càd corriger votre programme afin d'obtenir le fonctionnement attendu).
Nous utiliserons un peu plus tard Codeblocks, EDI gratuit sous Windows et linux, simple à installer et à utiliser.

Si vous souhaitez utiliser le compilateur en ligne gcc :
http://c.developpez.com/cours/mode-emploi-gcc/book1.php

```
gcc –W –Wall -o tp1 tp1.c
```

Pour utiliser des bibliothèques, la syntaxe est la suivante :

```
gcc -o binaire fichierSource.c -lbibliothèque
```

exemple :

```
gcc –W -Wall -o tp1  tp1.c -lm
```

lm pour utilisation de la librairie mathématique.

2. Travail à faire :

Les parties facultatives sont à réaliser par l'élève s'il a fini les parties précédentes dans la séance.

- **2.1 Première partie**

Sujet : Trouvez tous les nombres parfaits entre 2 et 10000.
Un nombre est parfait s'il est égal à la somme de ses diviseurs, lui non compris évidemment.
Exemple :

28 = 1 + 2 + 4 + 7 + 14 est donc parfait.

On affichera à l'écran les nombres parfaits trouvés.

Méthode : pour chaque nombre testé, faire une boucle faisant varier les diviseurs possibles. Tester pour chacun s'il est vraiment diviseur, et l'accumuler dans une variable "somme". Si le nombre est égal à "somme", alors il est parfait. On remarquera que tout nombre est divisible par 1.
Mesurez le temps de calcul en utilisant la fonction **clock** de la librairie **time.h** qui renvoie une approximation du temps d'utilisation du processeur depuis l'appel précédent à cette fonction :

```
float start =clock() ;
/* traitement…*/
printf("temps de calcul : %f \n ", (clock()-start) / CLOCKS_PER_SEC) ;
```

Rm :
Pour calculer x^y, on utilise la fonction de « math.h » pow(x,y) qui renvoie un double.
Pour calculer la racine carrée d'un nombre, on utilise la fonction de « math.h » sqrt(x) qui renvoie la racine carrée de x (type double).

- **2.2 Deuxième partie**

Sujet : Introduire la notion d'optimisation des calculs.
Méthode :
En arithmétique, on démontre que si on connaît les diviseurs d'un nombre N, d1, d2, d3, ... de valeurs inférieures à sa racine carrée, les diviseurs de valeurs supérieures s'en déduisent puisqu'ils s'écrivent N/d1, N/d2, N/d3...

Exemple : la racine carrée de 28 se situant entre 5 et 6, on peut écrire:

28 = 1 + 2 + 4 + 28/2 + 28/4 (28/1 étant exclu pour le calcul)

Reprendre le programme de la première partie en optimisant la boucle de variation des diviseurs.
L'exécution n'est-elle pas plus rapide ? Voyez-vous pourquoi ?

- **2.3 Troisième partie**

Sujet: Retour au nombre parfait.

Pour rechercher des nombres parfaits beaucoup plus grands (que le type int ou long ne permet), nous allons appliquer les formules ci-dessous.

Méthode:

Par hypothèse, si l'entier **a** =$\mathbf{2^{(n+1)}-1}$ est premier **avec n variant de 1 à 30**, alors le double **N = $\mathbf{2^n*(2^{(n+1)}-1)}$** est parfait.

Rechercher tous les nombres parfaits N en faisant varier l'exposant n de 1 à 30 dans a.

- ## 2.4 Quatrième partie

Sujet: Calcul du nombre de nombres premiers.

On veut calculer le nombre de nombres premiers existants jusqu'à un nombre <=n.

Méthode:

On saisit dans un premier temps une valeur n au clavier, ensuite on parcourt avec une variable x tous les nombres de 1 jusqu'à n : à chaque étape, si x est premier, on incrémente une variable somme.

Testez pour n=1 000 000, somme= 78498.
Pour n=10 000 000, somme= 664579.

- ## 2.5 Cinquième partie (facultatif)

Sujet: Décomposition fonctionnelle des traitements.

Reprenez les parties précédentes et faites réaliser les traitements non pas dans le main mais dans une fonction.

- ## 2.6 Sixième partie (facultatif)

Sujet: Compilation et exécution dans un terminal.

Pour ceux qui n'ont pas encore utiliser le compilateur en ligne, prenez un des programmes des parties précédentes, le compiler en ligne dans un terminal avec la commande « gcc » (voir exemple en début de sujet), et l'exécuter (pour exécuter un programme intitulé « monprogramme », on tape dans le terminal : « ./monprogramme »).