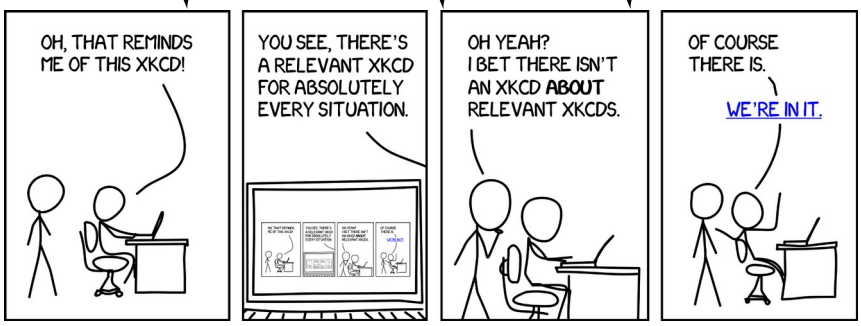
D A E M O N L A B

RECURSOR I



- DaemonLab -

Ce TP aborde la sujet des fonctions qui s'appellent elle-même.
Concernant la bande dessinée, xkcd, c'est bon, mangez en.

INDEX

01 – Avant-propos

Votre travail doit être rendu via votre bibliothèque, **libstd**.

Si vous faites erreur et que le dossier que vous utilisez pour votre rendu est différent, vous ne serez pas évalué faute d'avoir pu trouver votre travail.

Ce travail est a effectuer seul. Vous pouvez bien sur échanger avec vos camarades, néanmoins vous devez être l'auteur de votre travail. Utiliser le code d'un autre, c'est **tricher**. Et tricher annule **toutes** les médailles que vous avez reçu sur l'activité. La vérification de la triche est réalisée de la même manière que la correction : de manière **automatique**. Prenez garde si vous pensez pouvoir passer au travers.

Votre rendu doit respecter **strictement** l'ensemble des règles suivantes :

- Il ne doit contenir **aucun** fichier objet. (*.o)
- Il ne doit contenir **aucun** fichier tampon. (*~, #*#)
- Il ne doit pas contenir votre production finale (programme ou bibliothèque)

La présence d'un fichier interdit mettra immédiatement fin à votre évaluation.

Votre programme doit respecter les Tables de la Norme dans leur intégralité. Vous êtes invité à les observer depuis **l'Infosphère**. Elles sont disponibles comme ressource de cette activité.

02 – Fonctions autorisées

La bibliothèque logicielle venant avec le C est vaste et disponible. La LibLapin, que vous utilisez dans vos projets multimédia, est également vaste… Cependant nous avons fait le choix de vous interdire leurs utilisation intégrales, afin de vous amener progressivement à reprogrammer vous même ses fonctionnalités les plus utiles.

L'utilisation d'une fonction interdite est assimilée à de la triche. La triche provoque l'arrêt de l'évaluation et la perte des médailles.

Vous n'avez le droit d'utiliser aucune fonction issue de la LibC ou de la LibLapin à l'exception de celles que nous vous autoriserons explicitement.

Pouvoir utiliser une fonction ne signifie pas nécessairement que celle-ci soit utile à votre cas.

Pour cette activité, issu de la LibC, vous n'avez le droit qu'à la liste suivante :

- open
- close
- read
- srand
- cos
- atan2
- time
- write
- alloca
- atexit
- rand
- sin
- sqrt
- malloc, free

03 – Travail

Écrivez la fonction suivante :

```
int             std_recpow(double          x,
                           int             y)
```

Cette fonction fonctionne exactement comme la fonction **pow** de la bibliothèque standard ou comment votre **std_pow**, réalisé lors de votre période d'apprentissage. Il y a néanmoins une différence d'implémentation de taille.

Implémentez **std_recpow** sans utiliser de boucle. Les mot-clefs **white** et **do** sont interdits, en plus de l'habituelle interdiction de **for**. Vous n'avez pas le droit à **goto** non plus.

Comment faire ? Est ce impossible ? Non !
A quoi sert votre boucle ? A faire progresser un entier depuis 0 jusqu'à une certaine valeur. Pourriez vous à la place jouer avec un appel à la fonction **std_recpow** situé dans cette même fonction et avec ses paramètres ?

Attention spoiler : la réponse est oui. Comment ? L'appel a **recpow** depuis **recpow** n'a pas a se faire avec les mêmes paramètres…

Écrivez la fonction suivante :

```
int             std_recfactoriel(int       value);
```

De la même manière qu'avec **std_pow** et **std_recpow**, la fonction **std_recfactoriel** ne doit pas utiliser de boucle. Elle calcule la factorielle de **value**.

De la même manière qu'avec **std_factoriel**, si le nombre à calculer est trop grand, la fonction renverra -1 et mettra dans **errno** une valeur appropriée.

Écrivez la fonction suivante :

```
int             std_recfibonacci(int      value);
```

La fonction **std_recfibonacci** calcule la valeur numéro **value** dans la suite de Fibonacci caractérisée de la façon suivante :

f(x) = f(x – 1) + f(x – 2)
f(0) = 0
f(1) = 1

Cette fonction calculera la valeur en utilisation la récursion, c'est à dire en s'appelant elle-même. Cette façon de faire est dite naïve car elle comporte certains défauts. Lesquels ? Tentez de faire appel à un très grand nombre.

Écrivez la fonction suivante :

```
int             std_ackermann(int         m,
                              int         n);
```

Correspondant aux formules suivantes :

ackermann(m, n) =
- Si m = 0, n + 1
- Si m > 0 et n = 0, ackerman(m – 1, 1)
- Si m > 0 et n > 0, ackermann(m – 1, ackermann(m, n - 1))