Les défis de System Addict

Introduction

Cette page de défis vous permettra de vous casser la tête sur plusieurs défis, en sachant que la solution paraitra dans le numéro suivant. Les défis concernent diverses parties du numéro, et permettent d'acquérir des trucs et astuces, ou encore d'étudier des cas d'école. Ces cas d'école sont très riches en enseignements, car ils permettent de voir une petite astuce, et ensuite de l'appliquer quasi-automatiquement par la suite.
Le fait de se casser la tête sur ces défis vous permettra d'une part de vous entraîner sur des problèmes simples et concrets, et d'autre part de parfaire votre compréhension de certains articles du mag. Les auteurs restent à votre écoute, et sont joignables par mail (mais attention, ce ne sont pas des bêtes de sommes et ne peuvent pas tout savoir, ni même répondre à tout le monde). Allez, lancez vous, ça ne fait de mal à personne.

Défi 1

Le premier défi que l'on vous propose consiste à améliorer un petit programme sur deux points : d'une part sur l'algorithme employé, et d'autre part sur le code utilisé. Celui-ci est facilement optimisable, et permettra de gagner de la place en taille de code compilé, et de même en temps d'exécution.
Voici le programme à optimiser :

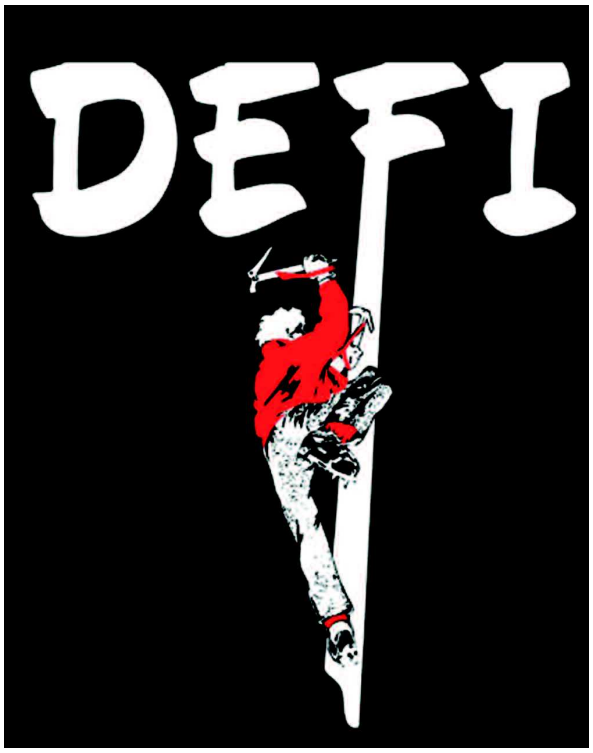
```
#include <stdio.h>
#include <stdlib.h>
#define MAX 1024

int main(int argc, char *argv[])
{
  char texte[MAX];
  char sortie[MAX];
  int a,t;

  //on demande d'entrer le texte
  printf("Entrez un texte :");
  scanf("%[^\n]s",texte);

  //puis on crée la chaine renversée
  a=0;
  t=strlen(texte);
  for(a=0;a<t;a++) sortie[a]=texte[t-a-1];
  sortie[a]='\0';

  //et on affiche la chaine renversee
  printf("Reversed : %s\n",sortie);

  system("PAUSE");
  return 0;
}
```

"Défis et solutions"

Cette section du mag est consacré à de petits défis, ou challenges, dont les solutions seront données au fur et à mesure des numéros. Ces défis sont pour la plupart techniques, et nécessite d'avoir assimilé plusieurs notions, voire même des connaissances externes. Mais ils sont aussi un excellent entrainement au développement et à l'administration, et permettent de s'exercer sur des cas concrets. C'est à vous de faire en sorte que cela fonctionne admirablement bien. Pas la peine d'envoyer de solutions au mag, ni même aux rédacteurs, car ils sont déjà débordés de boulot les pauvres.

Objectifs :

- Réduire la taille du code au maximum, afin d'obtenir un source clair et efficace.
- Modifier l'algorithme de manière à ce qu'il soit le plus optimal possible.
- Améliorer certains séquences afin de les rendre plus rapides.



Défi 2

Ce second défi est un peu plus complexe, car il vous demandera d'utiliser l'intégration de code assembleur dans un code en C vu dans l'article sur l'optimisation. Des connaissances en C et en assembleur x86 sont donc requises. Le but de ce défi est d'optimiser le fonctionnement d'un algorithme récursif. Petit rappel pour ceux qui ignorent le problème que posent les fonctions récursives en C : le passage des arguments à la fonction sont fait via la pile d'exécution, qui est en taille limitée. Cela limite donc le niveau d'imbrication de la fonction, qui se trouve ainsi limitée dans ses possibilités.En optimisant la gestion de la pile, et en arrangeant un peu l'algorithme, il y a moyen d'augmenter ces limites. Pour ce défi, vous devrez améliorer une fonction de calcul de factorielle :).

```
int factorielle(int n){
    if(n==0) return 1;
    else return n*factorielle(n-1);
}
```

Le but de ce défi est d'offrir à la fonction une pile assez importante. Il y a différentes méthodes pour y arriver, mais ce n'est pas si compliqué que cela. Et vous pouvez même à l'occasion optimiser un peu le code C de cette fonction. Bref, il y a de quoi s'amuser.

Des solutions seront proposées pour ces deux défis dans le prochain numréro, solutions détaillées avec code à l'appui et résultat opérationnels. Bon courage et creusez-vous les méninges !