CS-119(a) – ICC-C Série 13

2024-05-28

Important Les exercices 1-3 portent sur les fichiers. Les exercices 4-6 sont des exercices de révision, jetez-y un coup d'oeil aussi !

Exo1 SerDe

Inventez des noms et temps de parcours pour construire un tableau de quelques éléments du type

```
typedef struct info
{
    char *nom;
    int temps_sec;
} info_t;
```

Par exemple, `info_t data[] = {{"Alice", 11}, {"Alexandra", 13}, {"Antoine", 9}, {"Karim", 10}};`. Utilisez ensuite les fonctions `to_file` et `from_file` vues en cours. Avec `to_file` sauvegardez le tableau vers un fichier binaire. Rechargez-le en mémoire en vous aidant de `from_file` et assurez-vous que les données n'ont pas changé.

Exo2 Copier un fichier

Nous allons implémenter une fonction qui prend un nom de fichier source et un nom de fichier destination, et qui crée le fichier destination comme une copie exacte du fichier source.

```
long copy_file(const char *source_path, const char *dest_path);
```

La fonction retourne la taille en octets du fichier copié si l'opération réussit, et 0 sinon.

Il faut s'assurer que le fichier d'entrée existe - si ce n'est pas le cas, il faut retourner 0.

On ne sait pas à priori quelle est la taille du fichier qu'on veut copier, donc on va utiliser un emplacement de mémoire "tampon" où on lit des bouts successifs du premier fichier et on les écrit dans le deuxième. On peut par exemple utiliser un tableau de 100 caractères : `char buffer[100]`. Attention, le fichier n'aura probablement pas une taille qui est multiple de 100 octets !

Pour s'arrêter il faudra tester dans votre boucle (ou alors dans votre fonction récursive !) si on a atteint la fin du fichier avec la fonction `feof(.)`.

Vous pouvez tester si la copie a bien fonctionné avec l'utilitaire `diff` :

```
$ diff fichier_orig fichier_copie
```

ne devrait rien afficher si les deux fichiers sont identiques.

Attention aussi à ne pas écraser le fichier source... Assurez-vous d'avoir fait un backup du fichier que vous manipulez.

Exo3 (*) Message caché

Bob a caché un message dans un fichier binaire sur moodle. Ce fichier contient des entrées

```
typedef struct _garbled
{
    char c;
    long offset;
    int delta;
    int end;
} garbled_t;
```

Chaque élément correspond à un caractère du message (le champ `c`). Par contre les éléments du fichier ont été permutés et on ne peut lire ce fichier que si on connaît le "offset" du premier caractère.

Pour passer de l'élément `g(i)` à l'élément suivant `g(i+1)` dans le message, il suffit de "sauter" de `g(i).offset + S(i)` octets avec `fseek` juste après avoir lu `g(i)`. C'est un saut relatif à la position courante - avec `SEEK_CUR`. `S(i)` est un secret qui se calcule tout simplement en sommant les `g(j).delta` pour tous les `j <= i`.

Puisqu'il vous fait confiance, Bob partage la clé avec vous : le premier caractère du message se trouve à l'octet `50016` du fichier.

Déplacez-vous dans le fichier à cette position et affichez les caractères dans l'ordre original un par un. Pour tous les éléments sauf pour le dernier, le champ `g(i).end` vaut `0`. Pour le dernier élément, celui-ci vaut `1`.

Exo4 Stars

Qu'affiche ce code ?

```
    int v[6] = {5, 3, 0, 4, 2, 1};
    for (int *p = v; *p; p = v + *p)
    {
        printf("%d ", *p);
    }
    printf("\n");
```

Exo5 Bug mystère

Votre collègue a écrit le code suivant pour lister les `N` premiers nombres pairs positifs. Il a rencontré un problème en testant ce code et vous demande votre avis.

```
#include <stdio.h>

int* quelques_nombres_pairs(int n)
{
    int tableau[n];
    for (int i=0; i<n; i++)
    {
        tableau[i] = 2*i;
    }
    return tableau;
}

int main()
{
    int combien;

    scanf("%d", &combien);
    int *nombres_pairs = quelques_nombres_pairs(combien);

    for (int i=0; i<combien; i++)
    {
        printf("%d ", nombres_pairs[i]);
    }
    printf("\n");
}
```

Malheureusement, pour l'entrée `25` on voit s'afficher

```
0 32760 0 0 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48
```

Pourquoi ? Comment pourrait-on corriger ce code ?

Exo6 Odd

Souvenez-vous de la cellule d'une liste chaînée :

```
typedef struct _cell_t
{
    int contenu;
    struct _cell_t *next;
} noeud_t;
```

Qu'arrive-t-il à la liste `1 -> 2 -> 4 -> 4 -> 3 -> -5 -> -2` si on appelle la fonction suivante avec le premier élément de la liste en argument ?

```
void pr(cell_t *pcell)
{
    if (pcell == NULL || pcell->next == NULL)
    {
        return;
    }

    if (pcell->next->contenu % 2)
    {
        pr(pcell->next);
    }
    else
    {
        cell_t *a = pcell->next;
        pcell->next = pcell->next->next;
        free(a);

        pr(pcell);
    }
}
```