CS-119(a) – ICC-C Série 12

2024-05-13

Exo1 SerDe

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

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

Par exemple, `struct info 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.

Solution de l'exercice 1

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

void to_file(FILE *out, const struct info *record)
{
    int len = strlen(record->nom);
    fwrite(&len, sizeof(int), 1, out);
    fwrite(record->nom, sizeof(char), len, out);

    fwrite(&record->temps_sec, sizeof(int), 1, out);
}
```

```c
void from_file(FILE *in, struct info *record)
{
    int len;

    fread(&len, sizeof(int), 1, in);
    record->nom = malloc(len + 1);
    fread(record->nom, sizeof(char), len, in);
    record->nom[len] = 0; // End of string

    fread(&record->temps_sec, sizeof(int), 1, in);
}

int main()
{
    struct info data[] = {{"Alice", 11}, {"Alexandra", 13}, {"
   Antoine", 9}, {"Karim", 10}};
    struct info data_in;

    FILE *test_out = fopen("info_test", "w");
    for (int i = 0; i < 4; i++)
    {
        to_file(test_out, data + i);
    }
    fclose(test_out);

    FILE *test_in = fopen("info_test", "r");
    for (int i = 0; i < 4; i++)
    {
        from_file(test_in, &data_in);
        printf("%s - %d\n", data_in.nom, data_in.temps_sec);
        free(data_in.nom);
    }
    fclose(test_in);
}
```

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.

```c
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 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`. La commande suivante dans le terminal

```
$ 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.

Solution de l'exercice 2

```c
long copy_file(const char *source_path, const char *dest_path)
{
    FILE *src = fopen(source_path, "r");

    if (src == NULL)
    {
        return 0;
    }

    FILE *dest = fopen(dest_path, "w");
    long total_size = 0;
    char buffer[100];

    while (!feof(src))
    {
        long read_bytes = fread(buffer, 1, 100, src);
        fwrite(buffer, 1, read_bytes, dest);
        total_size += read_bytes;
    }

    fclose(src);
```

```
    fclose(dest);

    return total_size;
}
```

Exo3 (*) Message caché

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

```
struct garbled
{
    char c;
    long offset;
    int delta;
    int end;
};
```

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. Il faut donc se déplacer à cet endroit, lire `g(0)`, afficher le premier caractère, ensuite trouver `g(1)` à une distance de `g(0).offset + g(0).delta`, afficher le deuxième caractère, se déplacer à une distance `g(1).offset + g(0).delta + g(1).delta` pour trouver `g(2)`, etc.

Affichez tous les caractères dans l'ordre original un par un jusqu'à ce que vous ayez trouvé le dernier caractère. Pour tous les éléments sauf pour le dernier, le champ `g(i).end` vaut `0`. Pour le dernier élément, celui-ci vaut `1`.

Solution de l'exercice 3

```
#include <stdio.h>

struct garbled
{
    char c;
    long offset;
    int delta;
```

```
    int end;
};

int main()
{
    long start_offset = 0;

    scanf("%ld", &start_offset);

    FILE *f = fopen("garbled.msg", "r");

    fseek(f, start_offset, SEEK_SET);
    struct garbled g;
    int secret = 0;

    do
    {
        fread(&g, sizeof(struct garbled), 1, f);
        printf("%c", g.c);
        secret += g.delta;
        fseek(f, g.offset + secret, SEEK_CUR);
    }
    while (!g.end);

    fclose(f);
}
```