TP7 : manipulation de fichiers, modularisation du code

Algorithmique et projet de programmation — M1 mathématiques

15 novembre 2016

1 Commande `sort`

On rappelle qu'on essaye de recoder la commande `sort` des sytèmes UNIX, i.e. programme qui prend en paramètre un chemin d'accès à un fichier texte et qui affiche à l'écran l'ensemble de ses lignes triées lexicographiquement.

Lors du TP dernier, on a écrit deux fonctions :

```
char compare_str (char* s, char* t);
void sort_str (char** strs);
```

La première compare deux chaînes de caratère pour l'ordre lexicographique, la deuxième trie (selon votre méthode péférée) un tableau de chaînes de caractères.

Il s'agit maintenant d'utiliser ces fonctions pour coder un programme complet imitant `sort`. La bibliothèque `stdio.h` contient les fonctions suivantes afin de lire dans un fichier :

```
/*
  fopen takes the file path as a string and the opening mode that
  could be multiple thing: for our use, we just consider mode to be
  "r", opening the file in read-only mode.

  It returns a pointeur to a FILE structure, that we shall consider
  opaque.

  E.g.: FILE* f = fopen("path/to/my/file", "r");
*/
FILE* fopen (const char* path, const char* mode);

/*
  fgetc takes a pointer to a FILE and returns either a character just
  read in the file f or EOF, a constant meaning the End Of the File is
  reached.

  Beware that, when it does returns a character, it casts it to an
  int.

  E.g. : if( (c = fgetc(f)) != EOF ) printf("%c", (char) c);
*/
int fgetc (FILE* f);
```

```c
/*
  fgets takes a string s, a buffer length size and a pointer to a FILE
  f; it reads in f a string of length size-1 (stopping to the first
  '\n' or EOF if encountered) and puts it inside s. It writes a '\0'
  as the last character of s. It returns s if everything went right
  and NULL if not or if EOF is encountered before reading anything.

  E.g. : char s[10]; if ( fgets(s,10,f) != NULL ) printf("%s",s);
*/
char* fgets (char* s, int size, FILE* f);

/*
  rewind puts the current position in the file back to the beginning
  of the file pointed by f.

  It behaves like closing (see below) and reopening the file.
*/
void rewind (FILE* f);

/*
  fclose frees the memory allocated for the FILE structure pointed by
  f; it returns 0 upon success (and something else upon failure we are
  not interested in here).
*/
int fclose(FILE* f);
```

Exercice 1. Écrire un programme imitant la commande `sort`, qu'on appellera en ligne de commande comme suit :

```
./mysort path/to/my/file
```

où `path/to/my/file` est un chemin d'accès à un fichier texte.

En particulier, le programme commencera par déterminer le nombre de lignes et la taille maximale des lignes du fichier passé en paramètre pour allouera de la mémoire en quantité suffisante, avant d'appeler la fonction `sort_str`.

2 Votre première bibliothèque

Dans la suite, on va apprendre à *modulariser* son code. Jusqu'à maintenant, on a toujours mis tout notre code dans un même fichier : imaginer faire cela pour un projet de plusieurs milliers de ligne de code... Ce serait un bazar sans nom ! Heureusement, le langage `C` nous autorise à partitionner notre code en différents fichiers communiquant entre eux. En fait, vous utilisez déjà cette fonctionnalité du langage quand vous incluez les bibliothèques `stdio.h` et `stdlib.h`.

Au-delà de l'organisation, cette technique permet aussi d'*encapsuler* du code : c'est-à-dire de proposer à l'utilisateur (ici un autre codeur) une interface pour manipuler des données sans qu'il ait à en comprendre l'implémentation. Vous avez vous-même déjà expérimenté l'encapsulation en tant qu'utilisateur, pas plus tard qu'à l'exercice 1, dans la manipulation de fichier : vous n'avez pas besoin de connaître tous les détails de la structure `FILE` car vous avez accès à l'interface que sont les fonctions `fopen`, `fgets`, etc.

Enfin, c'est un bon moyen de maintenir une certaine *rétrocompatibilité*. Si demain les créateurs du langage `C` change l'implémentation de la structure `FILE` (pour s'adapter eux-même à un changement de norme dans les systèmes de fichiers par exemple), votre programme `sort` continuera de compiler sans souci du moment que les prototypes et comportements des fonctions `fopen`, `fgets`, etc. restent les mêmes. En revanche, si vous aviez utilisé les champs spécifiques de la structure `FILE`, votre programme pourrait tout à fait être rendu obsolète par une telle mise à jour des créateurs de `C`.

Il est temps de passer du côté développeur de l'encapsulation ! Vous avez sûrement remarqué que l'usage des chaînes de caractères en `C` est très pénible : tous les rouages de leur utilisations sont exposés à l'utilisateur, laissant libre cours à une gestion schyzophrénique de la mémoire, ce qui résulte généralement en tout un tas de `segfault`... On va donc créer une petite bibliothèque personnel, nommée `proper_string`, qui va gérer tous les problèmes de mémoire en arrière-plan et laisser à l'utilisateur seulement l'usage normale des chaînes de caractères.

Exercice 2. Créer un fichier `proper_string.c` contenant la déclaration d'une structure :

```c
struct string_s {
  int length; // maintain the length of the string
  char *val; // actual string, without the now useless trailing '\0'
} string_t;
// let's now call a proper string avery variable of type string_t
```

Écrire dans ce même fichier les fonctions de manipulation de chaînes de caractères que vous jugez utiles, e.g. :

```c
/* All functions get a 'ps_' prefix to recall that it is part of
     the 'proper_string' library' */
int ps_length (string_t str); // gives back the length of the
    proper string
string_t ps_init (const char* str); // transforms an actual C
    string into a proper string
string_t ps_empty (); // creates an empty proper string; should
    be equivalent to ps_init("")
string_t ps_copy (string_t str); // returns a copy of the proper
     string str
string_t ps_concat (string_t fst, string_t snd); // gives back
    the concatenation of two proper strings
void ps_print (string_t str); // prints the proper string
char *ps_toC (string_t str); // transform a proper string into a
     regular C string, i.e. a char array with trailing '\0'

/* ... whatever you could have wanted when using regular strings
     ...*/
```

On a donc dans le fichier `proper_string.c` une implémentation des chaînes de caractères et des fonctions de bases permettant de manipuler celles-ci. Ce n'est d'ailleurs pas la seule implémentaiton possible ! Vous pourriez avoir des raisons de choisir une autre structure de données pour le champ `val` : liste chaînées ou doublement chaînées, une table de hashage, etc. Ou bien vous pourriez vouloir ajouter d'autres champs : un entier `nb_spaces` pour le nombre d'espaces, un booléen

`all_num` pour savoir si tous les caractères sont numériques, etc. C'est à vous de voir, en fonction de l'utilisation que vous voulez en faire !

Il va falloir maintenant mettre en place l'interface, ce à quoi l'utilisateur aura accès : cela se fait par l'intermédiaire d'une *header*.

Exercice 3. Créer un fichier `proper_string.h` contenant les prototypes des fonctions et des structures auxquelles vous voulez donner accès à l'utilisateur.

Il faut alors penser à l'inclure dans `proper_string.c` : c'est-à-dire qu'il faut ajouter au préprocessing une nouvelle ligne

```
#include "proper_string.h" // beware, it is quotes, not chevrons
```

Remarque. Remarquez bien qu'il n'y a aucune fonction `main` dans le fichier `proper_string.c`. C'est normal car on ne veut pas exécuter directement du code de ce fichier, mais seulement s'en servir comme d'une banque de fonctions.

On va maintenant utiliser cette bibliothèque.

Exercice 4. Écrire un programme dans un fichier `prog.c` (vous pouvez bien sûr utiliser le nom que vous souhaitez pour le source) qui prend deux chaînes de caractères en paramètres en ligne de commande et qui en fait la concaténation sous forme de `string_t` et qui affiche sa longueur.

Il faudra en particulier inclure `proper_string.h`, et on compilera en rajoutant le fichier annexe `proper_string.c` en argument de `gcc` :

```
gcc -Wall -o myprog prog.c proper_string.c
```

3 Pour le 22 novembre

Exercice 5. Recoder la commande `sort` mais en utilisant votre nouvelle bibliothèque de strings. On n'hésitera pas à y ajouter les fonctions nécessaires.