Programmation II
Examen final

3 mai 2018

Aucun document autorisé. Durée 3h.
Page suivante se trouvent des prototypes de fonctions utilisables.
Dans tout code, en cas d'erreur (interne ou provenant d'une fonction de bibliothèque) on arrêtera le programme après avoir affiché un message explicite.

Exercice 1 : stockage d'un nombre

Expliquer comment un nombre (entier ou réel) est stocké dans un ordinateur. Vous pouvez en particulier détailler les différents formats selon les architectures matérielles, comment sont stockés les nombres négatifs, les noms des types en C selon la taille en bits des nombres, etc.

Exercice 2 : taille d'un fichier

Écrire une fonction `size_t taille(char *filename)` qui renvoie la taille (en octets) d'un fichier dont le nom est passé en paramètre. En cas d'erreur elle doit arrêter le programme.

Exercice 3 : triangle de Pascal

le triangle de Pascal est un tableau à deux dimensions triangulaire : la ligne numéro n possède $n+1$ cases. Ou, dit autrement, c'est un tableau dynamique nommé `C` tel que `C[i]` est un tableau dynamique de $i+1$ `int`. La valeur d'une case en ligne n colonne p est définie ainsi :

$$C[n][p] = \begin{cases} 1 & \text{si } n = 0 \text{ ou } n = p \\ C[n-1][p] + C[n-1][p-1] & \text{sinon.} \end{cases}$$

Écrire une fonction `int **pascal(int nblignes)` qui alloue un triangle de Pascal à `nblignes` lignes, le remplit selon la définition ci-dessus (il n'y a pas lieu d'utiliser une fonction récursive), et renvoie son adresse.

Exercice 4 : décompression de fichiers RLE

La compression RLE (*Run Length Encoding*) est une des méthodes les plus simples pour compresser des données (on peut l'utiliser pour du texte, des images...) même si elle est moins efficace que Lempel-Ziv (format `zip`), Huffman ou autres.

Le fichier codé (compressé) est un fichier binaire constitué d'une suite de `signed char` (entier sur un octet valant entre -128 et +127) dont certains sont des *multiplateurs* (qui donnent un nombre de répétitions). Le fichier décompressé sera un fichier binaire de `signed char` également.

L'algorithme de décodage est le suivant. Le premier caractère est toujours un multiplicateur

- S'il vaut $n > 0$ (positif) alors les n octets suivants sont écrits tels quels dans le fichier de sortie
- S'il vaut $n < 0$ (négatif) alors l'octet suivant est recopié $-n$ fois dans le fichier de sortie
- Il ne peut pas être 0

Ensuite si le fichier n'est pas terminé on recommence : l'octet suivant est un multiplicateur.

Exemple : Pour améliorer la lisibilité de cet exemple, la valeur décimale des nombres est écrite (telle que le programme `od -t d1` pourrait l'afficher) et les multiplicateurs sont écrits en gras. Bien sûr pour le stockage "réel" en machine ce n'est pas le cas : il s'agit de fichiers binaires (de taille 14 et 19 octets, respectivement).

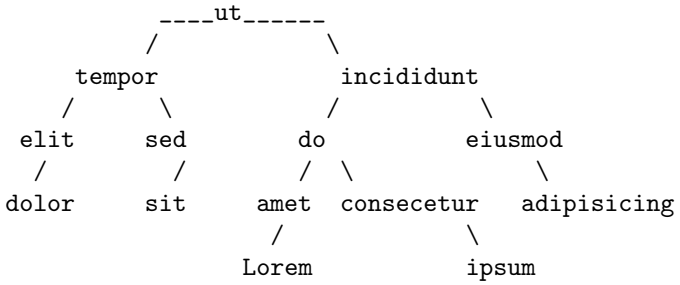
Compressé : **3** 10 -20 30 **-4** 8 **-7** 10 **5** 0 3 1 -12 24
Non-compressé : 10 -20 30 8 8 8 8 10 10 10 10 10 10 10 0 3 1 -12 24

Faire un programme qui décompresse un fichier codé sous format RLE. Il prend deux paramètres dans sa ligne de commande : le nom du fichier d'entrée (qui sera lu, au format RLE) et le nom du fichier de sortie (qui sera écrit, résultat de la décompression). Vous ferez attention au traitement d'erreurs en essayant de traiter le plus de cas possibles.

Exercice 5 : parcours d'un arbre binaire

1. Définir le type des nœuds d'un arbre binaire (chaque nœud a au plus deux fils) où chaque nœud stocke un mot (une chaîne de caractères de longueur quelconque).
2. Écrire une fonction qui renvoie la profondeur de l'arbre (ou du sous-arbre) dont on passe la racine comme unique paramètre. Elle doit renvoyer -1 si l'arbre est vide (racine `NULL`) et 0 si la racine est une feuille.
3. Faire une procédure qui prend en paramètre un nœud de l'arbre (on peut ajouter d'autres paramètres au besoin) et dont le résultat est l'affichage de l'ensemble des mots de l'arbre, écrits :
   – du niveau le plus bas jusqu'au plus haut (= la racine)
   – et (dans un même niveau) de gauche à droite

   et séparés par un espace. Par exemple pour l'arbre suivant (de racine `ut`)



   on affichera :

```
Lorem ipsum dolor sit amet consectetur adipisicing elit sed do eiusmod tempor incididunt ut
```

   Indication : Une façon de faire est d'afficher seulement les nœuds d'un niveau donné, grâce à une procédure auxilliaire, puis d'itérer par niveaux. On peut utiliser la fonction de la question précédente.

Fonctions utiles Ces fonctions sont suffisantes pour tous les exercices. Vous avez néanmoins le droit d'utiliser d'autres fonctions de la bibliothèque standard.

```
#include <stdio.h>
#include <stdlib.h>
#include <err.h>

void *calloc(size_t nmemb, size_t size);
void err(int codeRetour, const char *format, ...);  // ajoute message selon errno
void errx(int codeRetour, const char *format, ...); // n'ajoute pas message selon errno
int fclose(FILE *fd);
int feof(FILE *fd);
char *fgets(char *s, int size, FILE *fd); // lit au max size-1 caracteres et ajoute \0
FILE *fopen(const char *path, const char *mode); // mode = "r" "w" "a" "r+" "w+" ou "a+"
int fputs(const char *s, FILE *fd);
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *fd);
void free(void *ptr);
int fseek(FILE *fd, long offset, int whence); // whence = SEEK_SET SEEK_CUR ou SEEK_END
long ftell(FILE *fd);
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *fd);
void *malloc(size_t size);
int printf(const char *format, ...);
size_t strlen(const char *s);
```