Programmation en C et structures de données

guillaume.revy@univ-perp.fr

Tout ce qu'on n'a pas vu dans les autres TD

Exercice 1. Compter le nombre de caractères, mots et lignes d'un fichier

On souhaite ré-écrire une version simplifiée de la commande Unix `wc` (`man wc` pour plus de détails).

- **1.** Écrire une fonction qui prend en paramètre le nom d'un fichier, et qui détermine et renvoie le nombre de caractères, différents de l'espace, présent dans le fichier.
- **2.** Écrire un programme C qui permet d'utiliser cette fonction sur un fichier dont le nom est passé en ligne de commande.
- **3.** Modifier la fonction, et le programme, pour compter également le nombre de mots et de lignes.

Exercice 2. Découper un fichier trop volumineux

On souhaite maintenant écrire un programme qui permet de découper un fichier trop volumineux, en plusieurs fichiers, de taille maximale donnée.

- **1.** Écrire une fontion qui prend en paramètre le nom d'un fichier `input` et un entier k, et qui découpe le fichier en plusieurs `input.part_i` (avec $i \in \{0, 1, \cdots\}$) de taille au plus k octets.
- **2.** Écrire maintenant une fontion qui prend en paramètre le nom d'un fichier `input` et un nombre de partie, et qui reconstruit le fichier `input` initial à partir des fichiers `input.part_i`.
- **3.** Proposer un programme principal pour tester ces deux fonctions.

Exercice 3. Structure `une_date`

On souhaite modéliser une date, comprise entre le 01/01/1970 et le 31/12/2999.

- **1.** Définir une structure `une_date` contenant les attributs entiers suivant, dans cet ordre : les secondes (**`int`**), les minutes (**`long int`**), les heures (**`int`**), le jour (**`long int`**), le mois (**`int`**), et l'année (**`long int`**).
- **2.** Écrire un programme principal qui permet de créer une date, puis de lire et afficher les différents champs d'une date.
- **3.** Maintenant, sachant qu'en langage C, un entier de type **`int`** est représenté sur 32 bits (soit 4 octets), et un **`long int`** sur 64 bits (soit 8 octets), estimer la taille de la structure en mémoire.
- **4.** En utilisant la fonction **`sizeof`** (définie dans le langage C) , vérifier si cette estimation est correcte. Sinon, proposer une explication ?
- **5.** Comment réduire l'espace mémoire occupé par une structure de ce type ? Vérifier en pratique.

Exercice 4. Les unions en C et les champs de bits

En langage C, une union permet de rassembler des variables de types différents, mais qui occupent le même espace mémoire. Si ces membres sont des tailles différentes, l'espace réservé en mémoire correspond à celui de son plus grand élément. On va utiliser union pour décoder un nombre flottant.

On considère ici uniquement les nombres flottants, de type **`float`**, et définis de la manière suivante :

$$x = (-1)^s \times 1.m \times 2^{\mathrm{E}-127}, \tag{1}$$

avec $\rightsquigarrow$ le signe $s \in \{0, 1\}$,

$\rightsquigarrow$ la mantisse $m = m_1 m_2 \cdots m_{23}$ et $m_i \in \{0, 1\}, \forall i \in \{1, \cdots, 23\}$

$\rightsquigarrow$ l'exposant $\mathrm{E} = \mathrm{E_7E_6}\cdots\mathrm{E_0} \in \{0, \cdots, 127\}$ et $\mathrm{E_7}, \mathrm{E_6}, \cdots, \mathrm{E_0} \in \{0, 1\}$.

En machine, un nombre flottant de cette forme est représenté par un entier 32 bits, de la manière suivante.

s	$\mathrm{E_7}\ \mathrm{E_6}\cdots\mathrm{E_0}$	$m_1\ m_2\cdots m_{23}$

► **1.** Définir une union `float32_t`, permettant de représenter un nombre flottant de type **`float`**, et de lui associer sa représentation entière de type **`unsigned int`**.

► **2.** Quelle la représentation entière du nombre flottant le plus proche de 3.14. En utilisant des masques et des opérateurs bits-à-bits, déterminer les valeurs du signe s, de la mantisse m et de l'exposant E ? Est-ce cohérent avec la définition (1) ?

► **3.** Modifier l'union `float32_t` de telle sorte que le signe, la mantisse et l'exposant d'un nombre flottant puisse être déterminés directement, sans opérations bits-à-bits.

Exercice 5. Pile et file

Pour inverser l'ordre des éléments d'une pile, on peut utiliser une file comme stockage intermédiaire. Il suffit de dépiler tous les éléments de la pile, de les enfiler dans la file, puis de les ré-empiler dans l'ordre où ils ont été enfilés. On utilisera une liste chaînée comme structure de données sous-jacente.

► **1.** Définir les structures `liste`, `pile` et `file` permettant de définir une liste chaînée d'entiers, une pile et une file, respectivement.

► **2.** Écrire les fonctions `creer_pile` et `detruire_pile` qui permettent de créer une pile vide et de détruire une pile passée en paramètre.

► **3.** Écrire les fonctions `creer_file` et `detruire_file`.

► **4.** Écrire les fonctions `empiler`, `depiler` qui permettre d'empiler et dépiler un entier dans la pile.

► **5.** Écrire les fonctions `enfiler` et `defiler`.

► **6.** Écrire une fonction `inverser` qui inverse l'ordre des éléments d'une pile.

► **7.** Écrire les fonctions `afficher_pile` et `afficher_file`, et tester l'inversion.

Exercice 6. Evaluation d'expressions en notation polonaise inverse

La notation polonaise inverse permet d'écrire une expression arithmétique de manière non ambiguë sans utiliser de parenthèse. Par exemple, l'expression $3 \times (4 + 7)$ s'écrit de la manière suivante :

$$4\ \ 7\ +\ 3\ \times \quad \text{ou encore} \quad 3\ \ 4\ \ 7\ +\ \times.$$

On considère les expressions arithmétiques faisant intervenir des entiers sur un caractère **uniquement** et les opérateurs $\{+, -, \times, /\}$.
Dans ce formalisme :

— l'ordre des opérandes est conservé,
— les calculs se font en lisant l'expression de gauche à droite,
— et les calculs intermédiaires peuvent être gérés avec un pile.

Par exemple, considérons l'expression '3 4 7 + ×', et commençons avec une pile vide : $P = \{\}$. Ensuite l'évaluation de l'expression s'effectue de la manière suivante en lisant cette expression de gauche à droite :

1. on lit '3' et on l'empile dans P : $P = \{3\}$,
2. on lit '4' et on l'empile dans P : $P = \{3, 4\}$,
3. on lit '7' et on l'empile dans P : $P = \{3, 4, 7\}$,
4. on lit '+', on dépile donc $o_2 = 7$ et $o_1 = 4$ et on empile $o_1 + o_2 = 11$ dans P : $P = \{3, 11\}$,
5. et on lit '×', on dépile donc $o_2 = 11$ et $o_1 = 3$ et on empile $o_1 \times o_2 = 33$ dans P : $P = \{33\}$.

On utilisera les structures définies à l'exercice précédent.

► **1.** Écrire une programme qui lit une chaîne de caractères représentant une expression en notation polonaise inverse, qui parcourt cette chaîne de caractères pour évaluer l'expression et afficher son résultat.