TP 2 : champs de bits, `struct`, `union` et `enum`

Langage C (LC4)
Semaine du 8 février 2010

On rappelle que si `entier` est un **`int`**, alors sa valeur est affichée par `printf("%d", entier);`
On rappelle aussi que les entiers **`int`** sont codés sur 32 bits, et que les caractères **`char`** sont codés sur 8 bits. On a donc :

$$\begin{array}{cc} \text{unsigned int i} & 0 \leq x < 2^{32} \\ \text{unsigned char c} & 0 \leq c < 2^8 \end{array}$$

1 Simulation de tableaux

On va montrer qu'il est possible de stocker quatre **`unsigned char`** dans un **`unsigned int`**. Dans cette section, **on n'utilisera que les opérateurs logiques bit-à-bit** `&` (et), `|` (ou), `<<` (décalage à gauche, ce qui correspond à une multiplication par une puissance de 2) et `>>` (décalage à droite, division entière par une puissance de 2). Cependant, on pourra également utiliser l'incrémentation `++` ou de décrémentation `--` si on veut faire des boucles **`for`**).

Question 1. Écrire une fonction **`unsigned int`** `recomposer(`**`unsigned char`** `c[])` qui, étant donné un tableau c supposé de taille 4, calcule et renvoie l'entier :

$$c_3.(2^8)^3 + c_2.(2^8)^2 + c_1.(2^8)^1 + c_0.(2^8)^0$$

Indication : Si deux entiers non signés `a` et `b` ont des bits *disjoints* (i.e., si `a & b == 0`), alors on a `a + b == a | b`.

Question 2. Écrire une fonction **`void`** `decomposer(`**`unsigned int`** `i, `**`unsigned char`** `c[])` qui, étant donnés un entier i et un tableau c supposé de taille 4, écrit dans le tableau c les quatre entiers c_0, c_1, c_2, c_3 vérifiant :

$$\begin{cases} 0 \leq c_0, c_1, c_2, c_3 < 2^8 \\ i = c_3.(2^8)^3 + c_2.(2^8)^2 + c_1.(2^8)^1 + c_0.(2^8)^0 \end{cases}$$

Tester ces deux fonctions pour constater qu'elles sont réciproques l'une de l'autre.

2 Compositions de trains

Considérons une compagnie ferroviaire qui transporte aussi bien des voyageurs que des marchandises. Elle a donc besoin de trois types de matériel, appelés **éléments** :
– des **voitures** pour les voyageurs, caractérisées par le nombre de passagers embarqués,
– des **wagons** pour les marchandises, caractérisés par leur masse en tonnes (marchandise comprise) et le prix en euros payé par le client qui expédie la marchandise
– des **locomotives**.
Dans cette section, on n'oubliera pas de tester ses fonctions à chaque fois.

Question 3. Définir un type `element` pour représenter un élément de train, qui puisse être soit une locomotive, soit un wagon, soit une voiture. Dans le désordre, il faut trois **`struct`** et une **`union`**, et éventuellement un **`enum`**. Comment trouver le type d'un élément (i.e. savoir si un élément est une locomotive, un wagon ou une voiture) ?

Question 4. Écrire une fonction **`void`** `afficherElement(element e)` qui affiche à l'écran les caractéristiques d'un élement (on pourra utiliser un **`switch`**) :
– pour une locomotive, simplement « locomotive »
– pour un wagon, « wagon de ... tonnes valant ... euros »
– pour une voiture, « voiture de ... passagers »

Question 5. Un train est constitué d'un nombre quelconque de voitures, wagons et locomotives, dans n'importe quel ordre.
Cependant, comme les gares de la compagnie sont petites, un train comporte au maximum 10 éléments. Mais il peut en comporter moins. Définir un type `train` caractérisé par ses éléments et le nombre effectif d'éléments. Définir une fonction **`void`** `afficherTrain(train t)` qui affiche à l'écran le descriptif d'un train : nombre de ses éléments et caractéristiques de chaque élément.

Question 6. À la différence des wagons qui ont chacun leur propre masse, chaque locomotive ou voiture pèse 100 tonnes (on néglige la masse des passagers). Écrire une fonction **`int`** `masseElement(element e)` qui renvoie la masse d'un élément en tonnes. En déduire une fonction **`int`** `masseTrain(train t)` qui renvoie la masse d'un train.

Question 7. Un train qui comporte n locomotives peut rouler si sa masse totale (locomotives comprises) est inférieure à $500n$ tonnes. Écrire, **sans utiliser de multiplications**, une fonction **`int`** `estValide(train t)` qui renvoie 0 s'il n'y a pas assez de locomotives dans le train, et une valeur non nulle sinon.

Question 8. Le revenu d'un train se calcule le jour du départ. En effet, alors que les wagons marchandises sont réservés et payés longtemps à l'avance, les passagers paient leur billet au dernier moment, le tarif n'étant connu que le jour même du départ. Écrire une fonction **`int`** `revenuElement(element e, int prixBillet)` qui calcule le revenu en euros d'un élément suivant les contraintes :
– Une locomotive coûte 10 000 euros à la compagnie (revenu négatif)
– Le revenu d'un wagon est donné parmi ses caractéristiques
– Chaque passager d'une voiture paie `prixBillet` euros, mais la voiture elle-même coûte 2 000 euros à la compagnie (coût d'entretien, incompressible quel que soit le nombre de passagers, de sorte qu'une voiture est déficitaire si elle est insuffisamment remplie)

En déduire une fonction `int revenuTrain(train t, int prixBillet)` calculant le revenu d'un train entier.

Commentaire. En pratique, ce genre de fonction peut être utilisé par les analystes financiers de la compagnie ferroviaire pour déterminer le prix minimal d'un billet de train à partir de prévisions de fréquentation.

Question 9. Quelle est la taille mémoire minimale qu'occupe une donnée de type `element` ?

Question 10. Si on impose que :
- la masse d'un wagon soit strictement inférieure à 2^8 tonnes, et son revenu soit positif et strictement inférieur à 2^{22} euros
- une voiture admette strictement moins de 2^{30} passagers (ce qui est réaliste !)

alors un élément de train (qui peut être un wagon, une voiture ou une locomotive) peut-il être codé sur un entier de 32 bits (`unsigned int`) ? Si oui, trouver un codage et écrire une fonction `unsigned int coderElement(element e)` qui code un élément sous la forme d'un entier, puis réécrire la fonction `void afficherElementCode(unsigned int elementCode)` sur le modèle de `afficherElement` mais en prenant en paramètre un élément codé sous la forme d'un entier.