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

Langage C (LC4)
Semaine du 8 février 2010

1 Les nombres rationnels

Dans toute cette section, on utilisera le type **`int`** pour représenter les nombres entiers.

Question 1. Définir un type `rationnel` pour représenter un nombre rationnel donné par le numérateur (champ `numerateur`) et le dénominateur (champ `denominateur`). On ne demande pas que numérateur et dénominateur soient premiers entre eux.

Question 2. Définir une fonction `rationnel construireRationnel(int n, int d)` qui étant donné deux **`int`** représentant des entiers n et d, renvoie le rationnel n/d, et une fonction **`void`** `afficherRationnel(rationnel r)` affichant à l'écran le rationnel r sous la forme n/d, **en utilisant une seule fois `printf`**. On rappelle que si `x` est un **`int`**, alors `printf("%d", x);` affiche la valeur de `x`.

Question 3. Un élève donne le code suivant pour calculer l'opposé $(-n)/d$ d'un rationnel n/d :

```
void oppose(rationnel r) {
     r.numerateur = -r.numerateur;
}
```

Que peut-on dire de ce code ? Justifier la réponse et, le cas échéant, apporter les corrections nécessaires.

Question 4. Définir une fonction **`int`** `egaleRationnel(rationnel r1, rationnel r2)` qui renvoie 1 si les rationnels r_1 et r_2 sont égaux, et 0 sinon. (1 instruction)

Question 5. Définir une fonction `rationnel sommeRationnel(rationnel r1, rationnel r2)` qui calcule et renvoie la somme des deux rationnels donnés en argument (1 instruction). Quel problème pourrait survenir du fait que le numérateur et le dénominateur ne soient pas nécessairement premiers entre eux ?

2 Les booléens

C n'a pas de booléens. La condition d'un test **`if`**, d'une boucle **`while`**, etc. est un entier : 0 pour faux, et toute autre valeur non nulle (par défaut 1) pour vrai.
On donne le type de données suivant :

```
typedef enum {
        faux,
        vrai
} booleen;
```

Question 6. Cette définition convient-elle pour représenter les booléens de façon à pouvoir les utiliser tels quels dans un test pour **`if`**, **`while`**, etc. ?

Question 7. On essaie d'ajouter et de compiler le code suivant :

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    booleen b = 3;
    printf("%d\n", b);
    return EXIT_SUCCESS;
}
```

Ce code compile-t-il ? Si oui, qu'est-il affiché à l'écran ?

Question 8. Donner une autre façon équivalente de définir le type `booleen` (et donc les constantes `faux` et `vrai`) sans utiliser **`enum`**.

Question 9. On remplace **`int`** `egaleRationnel(rationnel r1, rationnel r2)` par `booleen egaleRationnel(rationnel r1, rationnel r2)`. Que faut-il modifier dans le corps de la fonction ?

3 Calcul exact ou approché, avec choix automatique

Dans cette section, on voudrait pouvoir représenter des *valeurs numériques* de deux façons possibles :
– de manière exacte sous la forme d'un rationnel
– de manière approchée sous la forme d'un flottant double précision
Cependant, on veut qu'une valeur numérique donnée ne soit représentée que d'une seule façon à la fois, ceci pour deux raisons :
– pour ne pas avoir à faire deux fois les calculs, une fois pour la valeur exacte et une fois pour la valeur approchée
– parce que la valeur exacte peut ne pas toujours exister
On définit le type suivant :

```
typedef union {
        rationnel exact;
        double    approche;
} num;
```

Question 10. On essaie d'ajouter et de compiler le code suivant :

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    rationnel r = construireRationnel(18, 1);
    num x;
    x.exact = r;
    x.approche = 42.1729;
    printf("%d\n", egaleRationnel(r, x.exact));
    return EXIT_SUCCESS;
}
```

Ce code compile-t-il ? Si oui, affiche-t-il nécessairement 1 à l'écran (justifier la réponse) ?

Question 11. Peut-on écrire une fonction `booleen estExact(num x)` qui renvoie `vrai` si le numérique `x` est représenté de façon exacte, et `faux` sinon ? Si oui, on considèrera dans toute la suite qu'on aura `typedef num numerique;` . Si non, comment remédier à ce problème ? Dans ce dernier cas, créer, en se servant du type `num`, un type `numerique` et une fonction `booleen estExact(numerique x)`.
Écrire une fonction `numerique construireNumeriqueExact(rationnel r)` qui crée un numérique par sa valeur exacte, et une fonction `numerique construireNumeriqueApproche(double v)` qui crée un numérique par sa valeur approchée.

Question 12. Écrire une fonction `void afficherNumerique(numerique nu)` qui affiche à l'écran le numérique passé en argument. Dans le cas où `nu` est représenté de façon approchée, on utilisera `printf("%f", x);` qui permet d'afficher à l'écran la valeur de `x` si `x` est un `double`.

Question 13. On donne la fonction ci-dessous permettant de calculer la valeur approchée d'un rationnel (elle est correcte, car C convertit automatiquement les entiers en flottants) :

```
double valeurApprocheeRationnel(rationnel r) {
        double n = r.numerateur;
        double d = r.denominateur;
        return (n / d);
}
```

Écrire une fonction `double valeurApprocheeNumerique(numerique nu)` qui renvoie la valeur approchée d'un numérique. En déduire une fonction `numerique sommeNumerique(numerique x, numerique y)` qui calcule la somme de deux numériques, de façon exacte si les deux sont représentés de façon exacte, et de façon approchée sinon.

4 Les sondages

Cette section est indépendante des trois autres.
Un organisme de sondage enquête sur les habitudes alimentaires des Français, et pose le questionnaire suivant :
– Mangez-vous de la tête de veau ?
– Buvez-vous du vin ?
– Mangez-vous du caviar ?
– Buvez-vous de l'eau minérale ?
Les sondés peuvent répondre oui ou non à chacune des quatre questions indépendamment. Le tableau ci-dessous indique les réponses de quatre sondés :

	Tête de veau	**Vin**	**Caviar**	**Eau minérale**
Jacques	oui	oui	non	oui
Nicolas	non	non	oui	oui
Jean-Louis	non	oui	non	non
Olivier	non	non	non	oui

Question 14. Si on représentait l'ensemble des réponses d'un individu avec une structure de quatre « booléens », quelle serait la taille minimale en mémoire ?

Question 15. On va montrer qu'en réalité, on peut stocker l'ensemble des réponses d'un individu dans un entier sur 4 bits, en effectuant des combinaisons booléennes bit-à-bit avec les opérateurs `&` (et) et `|` (ou).
Définir quatre constantes `teteDeVeau`, `vin`, `caviar` et `eauMinerale` telles que l'on puisse les combiner pour représenter l'ensemble des réponses d'un individu.

Question 16. Définir quatre variables `Jacques`, `Nicolas`, `JeanLouis` et `Olivier` pour représenter les réponses de Jacques, Nicolas, Jean-Louis et Olivier. Ce seront des **`char`** même si seulement 4 des 8 bits nous intéresseront ici.

Question 17. Définir une fonction **`char`** `teteDeVeauEtVin(`**`char`** `i)` qui renvoie une valeur non nulle si l'individu dont les réponses sont passées en paramètre mange de la tête de veau **et** boit du vin, et 0 sinon.