UGA - UFR IM^{2}AG **Master 2 CCI**

Programmation et Langages 1

Devoir surveillé du 05 novembre 2019

Durée : une heure – **Documents autorisés : une feuille A4 recto-verso + fiche de traduction Algo $\leftrightarrow$ C**

Exercice 1 ($\sim$ 4 points)

On considère le programme C ci-dessous :

```
void f1(int x) {
  x = x + 1;
}

void f2(int *x) {
   *x  = *x + 1 ;
}

int f3(int x) {
   int *y ;
   y = &x ;
   x = x + 1 ;
   return *y ;
}

int main() {
  int a, b ;
  a = 42 ;
  b = 1 ;
  f1 (a);
    /* (1) valeurs de a et b ? */
  f2 (&a) ;
    /* (2) valeurs de a et b ? */
  a = f3 (b);
    /* (3) valeurs de a et b ? */
  return 0 ;
}
```

Indiquez les valeurs des variables `a` et `b` aux points **(1)**, **(2)** et **(3)** indiqués en commentaire.

Exercice 2 (∼ 16 points)

On rappelle qu'en langage C les chaînes de caractères sont représentées dans des tableaux de caractères (de type `char`) et se terminent toujours par le caractère `'\0'` (marque de fin de chaîne).

On s'intéresse à deux manières de représenter un entier positif n en **base b**, où b désigne un entier entre 2 et 9 :

1. Le représentation nommée `Rep1` est un doublet contenant deux champs :
   — le champ `base` indique la base b dans laquelle l'entier n est exprimé ;
   — le champ `nbre` contient la chaîne de caractères représentant la valeur de n en base b.
2. La représentation `Rep2` est une chaîne de caractères de la forme `"b#xx...x"` où `b` indique la base b et `xx...x` la chaîne de caractères représentant la valeur de n dans cette base (le caractère `#` servant de séparateur).

Exemple : l'entier 6 en base 2 sera représenté par le couple `<2,"110">` selon `Rep1` et par la chaîne `"2#110"` selon `Rep2`.

Ces deux représentations sont définies par le lexique suivant :

L : la constante entière 20
Rep1 : le type <base : entier sur [2,9], nbre : tableau sur [0..L-1] de caractères>
Rep2 : le type tableau sur [0..L+1] de caractères

Q1 (2 points). Traduisez en C ce lexique.

Q2 (3 points). Traduisez en C la fonction suivante qui calcule la valeur décimale d'une valeur de type `Rep1`. Ainsi, `enDecimal(<2,"110">)` renvoie l'entier 6.

```
enDecimal : fonction (n1 : Rep1) → entier ≥ 0
lexique
      i, resultat : des entiers ≥ 0
debut
      i ← 0 ; resultat ← 0
      tantque n1.nbre(i) ≠ '\0'
            resultat ← resultat × n1.base + chiffre(n1.nbre(i))
            i ← i+1
      retourner resultat
fin
```

La fonction `chiffre` utilisée dans cet algorithme pour transformer le *caractère* `c` compris entre `'0'` et `'9'` en un *chiffre* entre 0 et 9 est donnée ci-dessous :

```
int chiffre (char c) {
      return c - '0'
}
```

Q3 (3 points). Une valeur `<b, "xx ...x">` de type `Rep1` est **correcte** si et seulement si :
- `b` est un entier entre 2 et 9 ;
- `"xx...x"` ne contient que des caractères représentant des chiffres entre 0 et `b`-1.

Ecrivez la fonction `estCorrecte (n1 : Rep1)` qui renvoie vrai si le paramètre `n1` est correct et faux sinon.

Q4 (4 points). Ecrivez en C l'action suivante qui transforme une valeur de type `Rep1` en la valeur de type `Rep2` correspondante.

enRep2 : action (la donnee n1 : Rep1, le resultat n2 : Rep2)
{e.i. : indifferent ;
e.f. : n2 est la représentation Rep2 de n1}

Indication : on pourra définir une fonction "réciproque" de la fonction `chiffre` fournie en **Q2**.

Q5 (4 points). Ecrivez en C la fonction suivante qui transforme une valeur de type `Rep2` en la valeur de type `Rep1` correspondante.

enRep1 : fonction (la donnee n2 : Rep2) $\rightarrow$ Rep1
{ enRep1(n2) est la représentation Rep1 de n2 }

Q6 (subsidiaire, hors-barême). Ecrivez en C la fonction `enBase` réciproque de la fonction `enDecimal`, qui prend en paramètre un entier n en base 10, une base b (entre 2 et 9), et renvoie la représentation Rep1 de n en base b. Ainsi, `enBase(6,2)` renvoie le couple `<2,"110">`.

Indication : le fragment d'algorithme suivant produit, à chaque itération, dans la variable x, les chiffres **de droite à gauche** [1] de la chaîne de caractères représentant l'entier n en base b :

tanque n $\neq$ 0
    x $\leftarrow$ n modulo b
    n $\leftarrow$ n div b

1. On pourra utiliser sans l'écrire une fonction qui produit l'image miroir d'une chaîne de caractères fournie en paramètre