Numération

- Ecriture d'un entier en base b.
- Lecture d'un entier en base b.
- Développement décimal d'un rationnel ; recherche d'un élément dans un tableau.

Ecrire un entier en base b

Pour obtenir, de droite à gauche, les chiffres d'un entier *n* en base *b*, il suffit d'écrire le reste de la division de *n* par *b*, de diviser *n* par *b*, et de recommencer tant que *n*>0. La fonction standard *printf* utilise cet algorithme.

```
int ecrire (long n, int chiffre[], int base) {
  int i;

  for (i = 0; n > 0; i++) {
    chiffre[i] = n % base;
    n = n / base;
  }
  return i;
}
```

Noter que si *a* et *b* sont entiers, *a/b* désigne, en C, le quotient entier, et *a%b* le reste de la division de *a* par *b*.

Cette fonction retourne le nombre de chiffres de l'écriture de *n* en base b. On peut l'utiliser comme suit, par exemple pour afficher les chiffres dans l'ordre usuel, c'est-à-dire en commençant par le chiffre de rang le plus élevé, et en terminant par celui des unités :

```
int i, n, p, c[32];
....
p = ecrire (n, c, 10);
for (i = p - 1; i >= 0; i--)
  printf (" %d", c[i]);
```

Si la base est supérieure à 10, cette fonction affiche des "chiffres" tels que 10, 11, 12... ; on peut sans difficulté corriger ce comportement, et modifier la fonction `ecrire` de telle sorte que les chiffres de *n* soient stockés dans une chaîne de caractères :

```
int ecrire (long n, char chiffre[], int base)
```

et utiliser les chiffres *a, b, c, ...* comme en numération hexadécimale.

Lire un entier en base b

Pour reconstruire un nombre de *p* chiffres, l'algorithme le plus efficace est le suivant :

```
long lire (int chiffre[], int p, int base) {
  int i;
  long n = 0;

  for (i = p - 1; i >= 0; i--)
    n = base * n + chiffre[i];
  return n;
}
```

Par exemple si les chiffres sont, de gauche à droite, 5, 7, 3 et 8 en base 10, les valeurs successives de *n* sont 0, 5, 57, 573 et 5738 ; noter qu'on a effectué seulement 4 multiplications (dont une par zéro), alors que le calcul de :

$$5738 = 5*10*10*10 + 7*10*10 + 3*10 + 8$$

semblait en demander 6.

La fonction *scanf* utilise cet algorithme, connu sous le nom de *schéma de Hörner*. Sa complexité est *linéaire* : le nombre d'opérations effectuées, en particulier de multiplications, est proportionnel au nombre de chiffres à lire. L'algorithme naïf, qui consiste à multiplier chaque chiffre par la base autant de fois que nécessaire, est de complexité *quadratique*, c'est-à-dire proportionnelle au *carré* du nombre de chiffres.

On peut modifier la fonction `lire` de telle sorte que les chiffres soient lus dans une chaîne de caractères (dont il est inutile de préciser la taille) :

```
long lire (char chiffre[], int base)
```

et utiliser les chiffres *a, b, c,* ... si la base est supérieure à 10.

Développement décimal d'un rationnel

Soit un rationnel *a/b* positif ; l'algorithme usuel de division, qui fournit les *n* premières décimales de la fraction, s'écrit en C :

```
for (i = 0; i <= n; i++) {
  q = a / b;
  r = a % b;
  a = 10 * r;
}
```

La partie entière de *a/b* et les *n* premières décimales apparaissent successivement dans la variable *q*. En fait ce développement est périodique, puisqu'il n'y a qu'un nombre fini possible de restes : la période est au plus *b*. Pour tenter de trouver cette période, lorsqu'elle n'est pas trop grande, on stocke chaque reste *r* dans un tableau nommé `reste`, et on examine si ce reste figure déjà dans ce tableau : si oui, on a trouvé la période, sinon on continue. Il ne faut pas oublier qu'un tableau est de taille limitée, et stopper l'algorithme lorsqu'on atteint cette limite ; dans ce dernier cas, la période est supérieure à la taille du tableau.

```
int periode;

int position (long t[], long x) {
  int i;

  for (i = 0; i < MAX && t[i] != x; i++);
  if (i < MAX)
    return i;
  else
    return -1;
}

int decimales (int t[], long a, long b) {
  int i;
  long r, reste[MAX];

  for (i = periode = 0; i < MAX && periode == 0; i++) {
    t[i] = a / b;
    r = a % b;
    a = 10 * r;
    reste[i] = r;
    periode =  i - position (reste, r);
  }
  return i;
```

```
}
```

Commentaires de programmation

- On a choisi d'enregister la période dans une variable globale, et de retourner le nombre de décimales ; on aurait pu choisir de retourner une structure, composée de la période et du nombre de décimales.

- La fonction `position` recherche une valeur *x* dans un tableau, et retourne son indice ; lorsqu'elle est appelée par la fonction `decimales`, *x* désigne le reste *r*, qu'on vient de placer justement en fin de tableau, donc on est certain que *x* figure dans le tableau ; mais il vaut mieux, et ce n'est pas difficile, écrire une fonction `position` générale, qui retourne -1 lorsque *x* ne figure pas dans le tableau ; cette valeur conventionnelle -1 est choisie car on ne risque pas de la confondre avec un indice.

- Dans la boucle
  ```
  for (i = 0; i < MAX && t[i] != x; i++);
  ```
  la définition de l'opérateur `&&` garantit que *t*[*i*] n'est pas évalué si l'indice *i* est invalide ; il est essentiel de ne pas intervertir les deux composantes du test.

- La même boucle se termine par un point-virgule, qui n'est pas une coquille, et indique que *le corps de la boucle est vide*. On peut aussi écrire la fonction `position` de la manière suivante, peut-être plus claire :

  ```
  int position (long t[], long x) {
    int i;

    for (i = 0; i < MAX; i++)
      if (t[i] == x)
        return i;
    return -1;
  }
  ```
  L'instruction `return` termine une fonction, même si ce n'est pas la dernière instruction. L'instruction `return -1` n'est donc exécutée que si *x* ne figure pas dans le tableau *t*.

- La double affectation `i = periode = 0` est correcte ; elle signifie `i = (periode = 0)`, et une affectation, en C, *est une expression* (dont la valeur est celle prise par la partie gauche de l'affectation après son exécution).

- C possède l'abréviation
  ```
  condition ? x : y
  ```
  pour désigner l'expression qui vaut *x* si la condition est vraie, *y* sinon. On peut donc récrire la dernière instruction de la fonction `position` :

  ```
  if (i < MAX)
    return i;
  else
    return -1;
  ```
  en une seule ligne :
  ```
  return i < MAX ? i : -1;
  ```

Voir aussi

Chaînes de caractères, développement en fraction continue.