ÉCOLE POLYTECHNIQUE **FILIÈRE MP**
OPTION INFORMATIQUE

CONCOURS D'ADMISSION 2002

COMPOSITION D'INFORMATIQUE

(Durée : 4 heures)

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.

Le langage de programmation choisi par le candidat doit être spécifié en tête de la copie.

On attachera une grande importance à la clarté, à la précision et à la concision de la rédaction.

⋆ ⋆ ⋆

Dans tout le problème un *système monétaire* est un ensemble de n entiers naturels non-nuls distincts $D = \{d_1, d_2, \ldots, d_n\}$, avec $n > 0$ et $d_n = 1$. Les d_i sont les *dénominations* du système. Par convention, les dénominations sont présentées par ordre *décroissant*. Par exemple, le système de l'euro est l'ensemble $\{500, 200, 100, 50, 20, 10, 5, 2, 1\}$.

Une *somme* (d'argent) est une suite finie $\langle e_1, e_2, \ldots, e_m \rangle$ d'entiers appartenant à D. Les éléments de cette suite sont des *espèces*. On remarquera bien que deux espèces d'une somme peuvent porter la même dénomination, qu'une somme peut être vide, et qu'il n'y a pas nécessairement d'espèces de dénomination 1 dans une somme. Par convention, les espèces sont présentées par ordre de dénominations décroissantes.

La *valeur* d'une somme S, notée $\mathcal{V}(S)$, est tout simplement la somme arithmétique de ses espèces, tandis que sa *taille*, notée $|S|$, est le nombre de ses éléments (l'entier m ci-dessus). Étant donné un entier naturel v, une somme de valeur v est un *représentant* de v. Par exemple, le portefeuille d'un citoyen européen peut contenir 3 billets de 10 euros, deux billets de 5 euros et une pièce de 1 euro. Cette somme est notée $\langle 10, 10, 10, 5, 5, 1 \rangle$, sa valeur est 41 et sa taille 6.

Une somme S est *extraite* d'une autre P dite portefeuille, si et seulement si S est une suite extraite de P. Intuitivement, « la somme S est extraite de P » signifie que l'on paye sa valeur à l'aide d'espèces prises dans le portefeuille P. Par exemple, notre citoyen européen peut payer exactement 15 euros en extrayant un billet de 10 euros et un autre de 5 euros de son portefeuille.

PROGRAMMATION. Dans les deux premières parties du problème, les systèmes monétaires et les sommes sont représentés en machine par des listes d'entiers.

(* *Caml* *)

```
type systeme == int list ;;
type somme   == int list ;;
```

{ *Pascal* }

```
type
   liste = ^cellule ;
   cellule = record
     contenu : integer ; suivant : liste ;
   end ;
   systeme = liste ;
   somme = liste ;
```

En outre, on supposera (pour les arguments) et on garantira (pour les résultats) les propriétés suivantes :
- tous les entiers présents dans les listes sont strictement positifs ;
- toutes les listes sont triées en ordre décroissant ;
- les listes de type `systeme` contiennent des entiers distincts ; leur dernier élément est 1.

En Pascal, la liste vide est `nil` et l'on pourra pour construire les listes utiliser la fonction suivante :

```
function cons(contenu : integer ; suivant : liste) : liste ;
var r : liste ;
begin
    new(r) ;
    r^.contenu := contenu ;
    r^.suivant := suivant ;
    cons := r
end ;
```

Cette fonction est applicable pour construire les listes des deux types `somme` et `systeme`.

Question 1. Écrire la fonction `valeur` qui prend une somme en argument et renvoie sa valeur.

(* *Caml* *)	{ *Pascal* }
`valeur : somme -> int`	`function valeur(s:somme) : integer ;`

I. Payer le compte exact.

Dans cette partie on considère le problème du paiement exact. Dans les termes du préambule, cela revient, étant donné un entier naturel p, à trouver un représentant de p.

Une démarche possible pour payer exactement le prix p est la démarche dite gloutonne, que l'on peut décrire informellement ainsi :
- donner l'espèce la plus élevée possible — c'est à dire de la plus grande dénomination d disponible et telle que $d \leqslant p$;
- recommencer en enlevant l'espèce donnée au prix à payer — c'est dire poser p égal à $p - d$.

Évidemment, le processus s'arrête lorsque le prix initial est entièrement payé.

Question 2. Dans cette question, on suppose que l'acheteur dispose toujours des espèces dont il a besoin.

a) Montrer que la démarche gloutonne réussit toujours.

b) Écrire une fonction `glouton` qui prend en argument un système monétaire `sys` et un prix à payer `p` et renvoie la liste des espèces à utiliser pour payer. La sommme renvoyée sera calculée en suivant la démarche gloutonne.

(* *Caml* *)	{ *Pascal* }
`glouton : systeme -> int -> somme`	`function glouton (sys:systeme ; p:integer):somme ;`

Question 3. On tient cette fois compte des ressources de l'acheteur. Dans les termes du préambule cela revient à trouver une somme ayant pour valeur le prix à payer et extraite d'une somme donnée, dite portefeuille.

a) Montrer, à l'aide d'un exemple utilisant le système européen, que la stratégie gloutonne peut échouer pour un prix donné, même si il est possible de payer compte tenu des ressources disponibles.

b) Écrire une fonction `paye_glouton` qui prend en argument une somme `pf`, représentant le contenu du portefeuille, ainsi qu'un prix `p` ; et qui renvoie une somme extraite de `pf` et dont la valeur est `p`. La somme renvoyée sera calculée en suivant la démarche gloutonne, si cette démarche échoue, la fonction `paye_glouton` doit renvoyer la liste vide.

(* *Caml* *)
```
paye_glouton : somme -> int -> somme
```

{ *Pascal* }
```
function paye_glouton
  (pf:somme ; p:integer):somme ;
```

Question 4. Écrire une fonction `compte_paiements` qui prend en arguments un portefeuille `pf` et un prix `p` ; et qui renvoie le nombre de façons de payer `p` à l'aide des espèces de `pf`. Autrement dit cette fonction renvoie le cardinal de l'ensemble des représentants de `p` extraits de `pf`.

(* *Caml* *)
```
compte_paiements : somme -> int -> int
```

{ *Pascal* }
```
function compte_paiements
  (pf:somme ; p:integer):integer ;
```

REMARQUE. Deux espèces de même dénomination sont distinguées. Ainsi, si le portefeuille est $\langle 2, 2, 2\rangle$ et le prix 2, alors il y a trois façons de payer.

II. Payer le compte exact et optimal.

Une somme est *optimale* lorsque sa taille est minimale parmi un ensemble de sommes de valeur donnée. Par exemple, la somme $S = \langle 20, 20, 1\rangle$ montre que le portefeuille de notre citoyen européen ($P = \langle 10, 10, 10, 5, 5, 1\rangle$) n'est pas optimal parmi les sommes de valeur 41. En effet, la taille de S est 3 et elle est strictement inférieure à la taille de P.

Dans cette partie un portefeuille P est fixé. Pour tout entier naturel p, on pose $M(p)$ égal à un représentant de p optimal parmi les représentants de p extraits de P, si une telle somme existe ; ou la somme vide, si une telle somme n'existe pas. Le but de cette partie est la détermination de $M(p)$.

Question 5. Le but de cette question préalable est de préciser quelques opérations sur les sommes.
a) Soit une somme S comprenant k espèces de dénomination d, on définit l'ajout de d à S, noté $S + \langle d\rangle$, comme la somme qui comprend $k + 1$ espèces de dénomination d et qui est inchangée autrement. Écrire la fonction `ajoute` qui prend en arguments une somme `s` et une dénomination `d` et qui renvoie la liste représentant l'ajout de `d` à `s`.

(* *Caml* *)
```
ajoute : somme -> int -> somme
```

{ *Pascal* }
```
function ajoute
  (s:somme ; d:integer):somme ;
```

b) On définit la différence de deux sommes S et S', notée $S - S'$, comme suit. Pour toute dénomination d, soit k le nombre d'espèces de dénomination d comprises dans S et k' le nombre d'espèces de dénominations d comprises dans S' :
- si $k > k'$, alors $S - S'$ comprend $k - k'$ espèces de dénomination d ;
- sinon, $S - S'$ ne comprend aucune espèce de dénomination d.

Écrire la fonction `diff` qui prend deux sommes en arguments et renvoie leur différence.

(* *Caml* *)
```
diff : somme -> somme -> somme
```

{ *Pascal* }
```
function diff(s,t:somme):somme ;
```

Question 6. Soit i un entier naturel. On note $T(i)$ l'ensemble des entiers naturels p, tels que la somme $M(p)$ est de taille i. On définit également une suite de fonctions M_0, M_1,...où la fonction M_i est la restriction à $\bigcup_{0 \leqslant j \leqslant i} T(j)$ de la fonction M.

Déterminer $T(i)$ et M_i dans le cas du portefeuille européen $\langle 10, 10, 10, 5, 5, 1\rangle$ et pour i égal à 0, 1 et 2.

Question 7. On suppose donné un tableau global de sommes, `tab`, de taille suffisante (cette condition est précisée par la suite) et dont chaque case vaut initialement la liste vide.

(* *Caml* *)	{ *Pascal* }
`tab : somme vect`	`tab : array [0..QMAX] of somme`

Pour un entier i donné, on encode simultanément l'ensemble $T(i)$ et la fonction M_i de la façon suivante :

- une liste d'entiers représente $T(i)$;
- si $M_i(p)$ est défini, alors la case d'indice p de `tab` contient $M_i(p) = M(p)$. Sinon, la case d'indice p de `tab` n'existe pas ou contient la liste vide.

a) Écrire la fonction `etape` qui prend en arguments, un portefeuille `pf`, un prix à payer `p` et une liste d'entiers représentant l'ensemble $T(i)$, et qui renvoie une liste d'entiers représentant $T(i+1)$. On supposera en outre que le tableau `tab` encode M_i avant l'appel de la fonction `etape` et on demande que le tableau `tab` encode M_{i+1} au retour de la fonction `etape`.

(* *Caml* *)	{ *Pascal* }

`etape :`
`  somme -> int -> int list`
`  -> int list`
	`function etape`
	`  (pf:somme ; p:integer ; ti:liste):liste ;`

On notera :

- les listes représentant $T(i)$ et $T(i+1)$ ne sont pas nécessairement triées ;
- la condition « `tab` est de taille suffisante » est précisée : le tableau `tab` est supposé tel qu'il existe bien des cases d'indice q, pour q inférieur ou égal à `p`.

b) En déduire une fonction `optimal` qui prend en argument un portefeuille `pf` et un prix `p` et qui renvoie une somme optimale de valeur `p` et dont les espèces sont extraites de `pf`. Si il n'est pas possible de faire l'appoint, la fonction `optimal` renverra la liste vide.

(* *Caml* *)	{ *Pascal* }
`optimal : somme -> int -> somme`	`function optimal(pf:somme ; p:integer):somme ;`

III. Étude des systèmes monétaires.

Un système monétaire est dit *canonique*, lorsque la stratégie gloutonne sans limitation de ressources (cf. question 2) appliquée à tout prix p produit une somme optimale parmi les représentants de p.

Question 8. Montrer que l'ancien système britannique $\langle 240, 60, 30, 24, 12, 6, 3, 1\rangle$ n'est pas canonique.

Le but de cette partie est de produire un programme qui décide si un système monétaire est canonique. Dans cette étude on fixe un système monétaire D de n dénominations, représenté cette fois par le vecteur

de n entiers naturels $D = (d_1, d_2, \ldots, d_n)$. Une somme S sera également représentée par un vecteur de n entiers naturels, $S = (s_1, s_2, \ldots, s_n)$, *mais s_i est cette fois le nombre d'espèces de dénomination d_i présentes dans la somme S.*

Ainsi le système européen est le vecteur $(500, 200, 100, 50, 20, 10, 5, 2, 1)$ et le portefeuille du citoyen est le vecteur $(0, 0, 0, 0, 0, 3, 2, 0, 1)$. Les définitions (et les notations) de la valeur et de la taille sont inchangées :

$$\mathcal{V}(S) = \sum_{i=1}^{n} s_i \cdot d_i, \quad |S| = \sum_{i=1}^{n} s_i$$

Par ailleurs les sommes sont ordonnées (totalement) selon l'ordre lexicographique, noté $<_\ell$ et défini ainsi : pour tous vecteurs U et V, on a $U <_\ell V$ si et seulement si :

1. il existe i, $1 \leqslant i \leqslant n$, tel que $u_i < v_i$;
2. et, pour tout j, $1 \leqslant j < i$, on a $u_j = v_j$.

Ainsi on a par exemple $(0,1) <_\ell (0,2)$ $(i = 2)$ et $(0,4) <_\ell (1,0)$ $(i = 1)$. On note $\leqslant_\ell$ la relation d'ordre définie par $U \leqslant_\ell V$, si et seulement si $U <_\ell V$ ou $U = V$.

On définit un second ordre total sur l'ensemble des sommes, noté $\sqsubseteq$, de la façon suivante :

$$U \sqsubseteq V, \quad \text{si et seulement si} \quad |U| > |V| \text{ ou } (|U| = |V| \text{ et } U \leqslant_\ell V)$$

Les relations d'ordre introduites permettent les *définitions* suivantes des représentants gloutons et optimaux (il n'est pas demandé de comparer ces nouvelles définitions aux anciennes). Étant donné un entier naturel p, le représentant glouton de p, noté $G(p)$ est le plus grand selon l'ordre lexicographique $\leqslant_\ell$ des représentants de p. Tandis que le représentant optimal de p, noté $M(p)$, est le plus grand selon l'ordre $\sqsubseteq$ des représentants de p.

Dès lors le système D est canonique, si et seulement si on a $G(p) = M(p)$ pour tout entier naturel p. En revanche, D n'est pas canonique, si et seulement si il existe un ou des entiers naturels w, dits *contre-exemples*, tels que $M(w) \neq G(w)$, c'est à dire tels que $M(w) <_\ell G(w)$.

PROGRAMMATION. Dans cette partie on considère l'entier n (nombre de dénominations du système monétaire) fixé. Les systèmes monétaires et les sommes sont représentées en machine par des tableaux d'entiers.

(Caml *)*

```
let n = ... (* n:int *) ;;
type tsysteme == int vect ;;
type tsomme == int vect ;;
```

{ Pascal }

```
const
   n = ... ;
type
   tsysteme = array [1..n] of integer ;
   tsomme   = array [1..n] of integer ;
```

(En Caml, on supposera ces tableaux de taille $n + 1$, de sorte que les indices des tableaux correspondent à ceux des vecteurs.)

Question 9. Écrire la fonction `tglouton` qui prend en argument un système monétaire `sys` selon le nouveau type et un prix `p` et qui renvoie le représentant glouton de ce prix. Le coût de `tglouton` (c'est-à-dire le nombre maximum d'instructions élémentaires utilisées lors d'un appel à `tglouton`) doit être linéaire en n.

(Caml *)*

```
tglouton :
   tsysteme -> int -> tsomme
```

{ Pascal }

```
function tglouton
  (sys:tsysteme ; p:integer) : tsomme;
```

Question 10.
a) Montrer que $p < q$ entraîne $G(p) <_\ell G(q)$.

Soit k, indice, avec $1 \leqslant k \leqslant n$. On pose I_k égal à la somme composée d'une seule espèce de dénomination d_k. Soit encore p, entier naturel.
b) On suppose que $G(p)$ comprend au moins une espèce de dénomination d_k. Montrer qu'alors on a $G(p - d_k) = G(p) - I_k$. (Le deuxième « $-$ » ci-dessus est la différence des sommes, que l'on peut simplement voir comme la soustraction appliquée aux vecteurs et définie composante par composante.)
c) De même, si p est tel que $M(p)$ comprend au moins une espèce de dénomination d_k, montrer que l'on a alors $M(p - d_k) = M(p) - I_k$.

Question 11. Dans cette question on suppose le système monétaire D non-canonique et on considère le contre-exemple minimal w, c'est à dire l'entier w tel que :
- $M(w) \neq G(w)$ (et donc $M(w) <_\ell G(w)$) ;
- et, pour tout entier $w' > w$, $M(w') = G(w')$.

On note $M(w) = (m_1, m_2, \ldots, m_n)$. Soient encore i l'indice minimal tel que $m_i > 0$ et j l'indice maximal tel que $m_j > 0$.
a) On note $G(w) = (g_1, g_2, \ldots, g_n)$. Montrer qu'il n'existe pas d'indice k, tel que $m_k > 0$ et $g_k > 0$.
b) Montrer que l'on a $i > 1$.
c) Montrer que l'on a $d_{i-1} < w < d_{i-1} + d_j$. monétaire,

Question 12. Toujours en supposant le système D non-canonique et en conservant les définitions et notations de la question précédente, on admet l'encadrement suivant (qui cette fois s'applique aux sommes) :

$$M(w) - I_j \leqslant_\ell G(d_{i-1} - 1) <_\ell M(w).$$

a) Montrer que cet encadrement permet de connaître les composantes de $M(w)$ en fonction de celles de $G(d_{i-1} - 1)$, en supposant i et j connus.
b) Écrire une fonction `canonique`, qui prend en argument un système monétaire `sys` et décide si ce système est canonique.

(* *Caml* *)	{ *Pascal* }
`canonique : tsysteme -> bool`	`function canonique(sys:tsysteme) : boolean`

Le coût de `canonique` doit être en $O(n^3)$.
c) Montrer que le système européen est canonique.

* *
*