ÉCOLE NATIONALE DES PONTS ET CHAUSSÉES,
ÉCOLES NATIONALES SUPÉRIEURES DE L'AÉRONAUTIQUE ET DE L'ESPACE,
DES TECHNIQUES AVANCÉES, DES TÉLÉCOMMUNICATIONS,
DES MINES DE PARIS, DES MINES DE SAINT-ÉTIENNE, DES MINES DE NANCY,
DES TÉLÉCOMMUNICATIONS DE BRETAGNE,
ÉCOLE POLYTECHNIQUE
(Filière T.S.I.)

CONCOURS D'ADMISSION 1998

ÉPREUVE D'INFORMATIQUE

Filière MP
(Durée de l'épreuve : 3 heures)

Les candidats et les candidates sont priés de mentionner de façon apparente sur la première page de la copie :
« INFORMATIQUE - Filière MP »

RECOMMANDATIONS AUX CANDIDATS ET CANDIDATES

L'énoncé de cette épreuve, y compris cette page de garde, comporte 12 pages.

Si, au cours de l'épreuve, un candidat ou une candidate repère ce qui lui semble être une erreur d'énoncé, il ou elle le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il ou elle a décidé à prendre.

Tout résultat fourni dans l'énoncé peut être utilisé pour les questions ultérieures même s'il n'a pas été démontré. Il ne faut pas hésiter à formuler les commentaires qui semblent pertinents même lorsque l'énoncé ne le demande pas explicitement.

COMPOSITION DE L'ÉPREUVE

L'épreuve comprend **trois problèmes indépendants.**

- **Premier problème :** le problème de logique, le problème n° 1 page 2, à résoudre en 60 minutes environ.
- **Deuxième problème :** le problème de théorie des automates, le problème n° 2 page 4, à résoudre en 60 minutes environ.
- **Troisième problème :** un problème de programmation au choix parmi deux, à résoudre en 60 minutes environ. **Attention !** Pour le troisième problème, un seul choix est possible. Le candidat ou la candidate précisera son choix sur sa copie et seul le problème correspondant sera corrigé. Les deux choix possibles sont :
  - le problème de programmation en langage Pascal, le problème n° 3 page 7, pour les candidats et les candidates désirant choisir ce langage ;
  - le problème de programmation en langage Caml, le problème n° 4 page 10, pour les candidats et les candidates désirant choisir ce langage.

1 Problème de logique — 60 mn environ

L'objet de ce problème est la conception d'un circuit logique pour un afficheur de chiffres.

Rappels et notations

Soit $\mathcal{B} = \{0, 1\}$ l'ensemble contenant les entiers naturels 0 et 1. On définit les opérations binaires $+ : \mathcal{B}^2 \longrightarrow \mathcal{B}$ et $\cdot : \mathcal{B}^2 \longrightarrow \mathcal{B}$ par $x + y = max(x, y)$ et $x \cdot y = min(x, y)$ où $max(x, y)$ et $min(x, y)$ désignent respectivement le plus grand et le plus petit des deux entiers x et y. On définit l'opération unaire $- : \mathcal{B} \longrightarrow \mathcal{B}$ par $\overline{0} = 1$ et $\overline{1} = 0$. On admettra *sans les démontrer* les propriétés élémentaires de ces trois opérations: commutativité, associativité, éléments neutres et absorbants, diverses distributivités, etc.

Soit $n \geq 1$ un entier, soit $\mathcal{I} = \{i_1, \ldots, i_n\}$ un ensemble fini non vide et soient $e_{i_1}, \ldots, e_{i_n} \in \mathcal{B}$, on note:

$$\sum_{i \in \mathcal{I}} e_i = (e_{i_1} + \ldots + e_{i_n}) \quad \text{et} \quad \prod_{i \in \mathcal{I}} e_i = (e_{i_1} \cdot \ldots \cdot e_{i_n})$$

Soient $x_1, \ldots, x_n \in \mathcal{B}$ et $u_1, \ldots, u_n \in \mathcal{B}$, on note:

$$[x_1, \ldots, x_n]^{u_1, \ldots, u_n} = \sum_{i \in [1,n]} y_i \quad \text{où} \quad \forall i \in [1, n], y_i = \begin{cases} x_i \text{ si } u_i = 0 \\ \overline{x_i} \text{ si } u_i = 1 \end{cases}$$

Soit $\mathcal{A}$ un ensemble dénombrable de variables propositionnelles, on note $\mathcal{F}$ l'ensemble des *formules* contruites à partir des variables de $\mathcal{A}$, du connecteur unaire de négation noté $\neg$ et des connecteurs binaires de disjonction noté $\vee$ et de conjonction noté $\wedge$. Une *valuation* est une application $i : \mathcal{A} \longrightarrow \mathcal{B}$. Une valuation se prolonge en une *interprétation* $I_i : \mathcal{F} \longrightarrow \mathcal{B}$ définie par: si $x \in \mathcal{A}$ alors $I_i(x) = i(x)$; si $F, G \in \mathcal{F}$ alors $I_i(F \vee G) = I_i(F) + I_i(G)$, $I_i(F \wedge G) = I_i(F) \cdot I_i(G)$ et $I_i(\neg F) = \overline{I_i(F)}$. Soient F et G deux formules, si pour toute valuation $i : \mathcal{A} \longrightarrow \mathcal{B}$ on a $I_i(F) = I_i(G)$ alors on dit que F et G sont *logiquement équivalentes.*

Problème

On se donne $(x_k)_{k \in \mathbb{N}^*}$, une suite *particulière* de variables propositionnelles deux à deux distinctes. Pour tout entier $k \geq 1$ on note $\mathcal{F}_k$ le sous-ensemble de $\mathcal{F}$ composé des formules construites à partir des variables propositionnelles $x_1, \ldots, x_k$ et des connecteurs $\neg$, $\vee$ et $\wedge$.

1. Soit $k \geq 1$ et soit $F \in \mathcal{F}_k$; donner un procédé de construction inductif d'une fonction notée $\varphi_k^F : \mathcal{B}^k \longrightarrow \mathcal{B}$ telle que pour toute valuation $i : \mathcal{A} \longrightarrow \mathcal{B}$, on ait $I_i(F) = \varphi_k^F(i(x_1), ..., i(x_k))$. Prouver la validité de ce procédé.

2. Soit $k \geq 1$ et soit $f : \mathcal{B}^k \longrightarrow \mathcal{B}$ une application ; on pose:

$$\mathcal{B}_f^k = \left\{(u_1, \ldots, u_k) \in \mathcal{B}^k \text{ tels que } f(u_1, \ldots, u_k) = 0\right\}$$

Montrer que si $\mathcal{B}_f^k \neq \emptyset$ alors:

$$\forall (b_1, \ldots, b_k) \in \mathcal{B}^k, f(b_1, \ldots, b_k) = \prod_{(u_1, \ldots, u_k) \in \mathcal{B}_f^k} [b_1, \ldots, b_k]^{u_1, \ldots, u_k}$$

3. Soit $k \geq 1$ et soit $f : \mathcal{B}^k \longrightarrow \mathcal{B}$; déduire de la question 2 page ci-contre le procédé de construction d'une formule $F_f \in \mathcal{B}_k$ en forme normale conjonctive telle que $f = \varphi_k^{F_f}$.

4. Soit $k \geq 1$; montrer que pour toute formule $F \in \mathcal{F}_k$, il existe une formule $F' \in \mathcal{F}_k$ en forme normale conjonctive et logiquement équivalente à F.

5. Soit $k \geq 1$ et soit $f : \mathcal{B}^k \longrightarrow \mathcal{B}$; déduire de la réponse à la question 3 une méthode permettant de construire élémentairement *à partir du graphe* de f une formule $F \in \mathcal{F}_k$ en forme normale conjonctive et telle que $f = \varphi_k^F$. On conviendra de représenter le graphe de f par une table du type de celle donnée figure 1. Appliquer la méthode à la fonction f à 2 arguments définie par la table :

x_1	x_2	$f(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	0

FIG. 1 – *Graphe d'une fonction* $f : \mathcal{B}^2 \longrightarrow \mathcal{B}$.

6. On considère les nombres entiers de 0 à 7 codés en base 2. Chaque nombre $n \in [0,7]$ possède une représentation binaire $\overline{xyz}^2$ telle que $x, y, z \in \mathcal{B}$ et $n = 2^2.x + 2^1.y + 2^0.z$. On désire concevoir un afficheur digital comportant 7 segments lumineux numérotés s_1 à s_7. L'afficheur possède trois entrées logiques x, y et z correspondant à la représentation binaire d'un nombre $\overline{xyz}^2 \in [0,7]$. Il doit afficher le nombre en allumant les segments correspondants. La figure 2 montre une représentation d'un tel afficheur et la figure 3 montre les affichages des huit nombres qui nous intéressent.

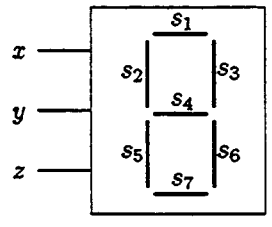

FIG. 2 – *Représentation d'un afficheur digital.*

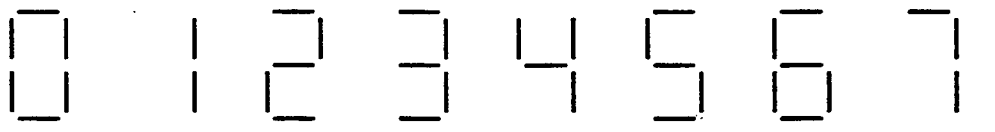

FIG. 3 – *L'affichage des nombres de* 0 *à* 7.

Exemple. Si $x = 1$, $y = 0$ et $z = 0$, l'afficheur doit afficher le nombre 4 correspondant à la représentation binaire $\overline{100}^2$. Il allumera donc uniquement les segments s_2, s_3, s_4 et s_6.

On désire exprimer le comportement de chaque segment s_i de l'afficheur pour $i \in [1,7]$ par une fonction $f_i : \mathcal{B}^3 \longrightarrow \mathcal{B}$ telle que $f_i(x,y,z) = 1$ si le segment s_i doit être allumé pour le nombre de représentation binaire $\overline{xyz}^2$, $f_i(x,y,z) = 0$ sinon. Donner les graphes en table des deux fonctions f_1 et f_7.

7. Donner les deux formules logiques $F_1 \in \mathcal{F}_3$ et $F_7 \in \mathcal{F}_3$ en forme normale conjonctive telles que $\varphi_3^{F_1} = f_1$ et $\varphi_3^{F_7} = f_7$.

8. On se donne les portes logiques et le nœud *duplicateur* de la figure 4. Le nœud duplicateur prend une entrée x et la reproduit sur ses deux sorties. On suppose connu le fonctionnement de ces éléments de circuits logiques. En utilisant la question 7, dessiner le circuit logique réalisant l'allumage du segment s_7 en fonction des trois entrées x, y et z.

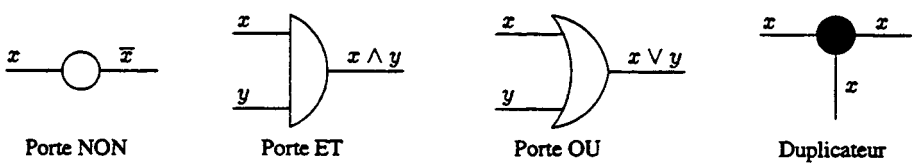

FIG. 4 – *Les portes logiques.*

FIN DU PROBLÈME DE LOGIQUE

2 Problème sur les automates — 60 mn environ

Nous nous proposons de démontrer dans ce problème que l'ensemble des mots dont une puissance k-ième quelconque fixée est dans un langage reconnaissable est lui-même reconnaissable.

Rappels pour fixer les notations

On appelle *alphabet* un ensemble fini A dont les éléments sont appelés *lettres*. On note A^* l'ensemble des *suites finies* de lettres de A, que l'on appelle *mots*. La *longueur* d'un mot f est le nombre de lettres qui le composent et est notée $|f|$; par exemple, $A = \{a,b\}$, $f = aba$, $|f| = 3$. La suite vide est notée 1_{A^*}. Si $k \in \mathbb{N}$ et $f \in A^*$, on note f^k le mot obtenu en répétant k fois consécutivement le mot f. Un *langage* de A^* est une partie de A^*.

Un *automate fini déterministe* $\mathcal{A} = <Q, A, \delta, i, T>$ est une structure telle que Q est un ensemble *fini* d'éléments appelés *états* de $\mathcal{A}$; l'élément i de Q est *l'état initial* de $\mathcal{A}$; T, sous-ensemble de Q, est l'ensemble des *états terminaux* de $\mathcal{A}$; et δ, fonction *partiellement définie* de $Q \times A$ dans Q est la *fonction de transition* de $\mathcal{A}$. Chaque élément $(p, a, \delta(p,a))$ du graphe de δ est appelé une *transition* de $\mathcal{A}$ dont *l'étiquette* est a.

On étend δ en une fonction *partiellement définie* de $Q \times A^*$ dans Q, encore notée δ, par :

$$\forall p \in Q \quad \delta(p, 1_{A^*}) = p \ , \quad \forall p \in Q \, , \, \forall a \in A \, , \, \forall f \in A^* \quad \delta(p, fa) = \delta(\delta(p,f), a)$$

Pour alléger l'écriture, et s'il n'y a pas d'ambiguïté, on note $q = p \cdot f$ au lieu de $q = \delta(p,f)$.

Un mot f de A^* est *reconnu* par $\mathcal{A}$ si $i \cdot f \in T$. Le *langage reconnu* par $\mathcal{A}$, noté $L(\mathcal{A})$, est l'ensemble des mots reconnus par $\mathcal{A}$, c'est-à-dire $L(\mathcal{A}) = \{f \in A^* \mid i \cdot f \in T\}$. Deux automates sont *équivalents* s'ils reconnaissent le même langage.

On représente graphiquement un automate comme dans l'exemple de la figure 5 : un état est figuré par un cercle marqué par son nom, une transition par une flèche à proximité de laquelle on trouve son étiquette ; l'état initial est marqué d'une flèche entrante tandis que les états terminaux sont marqués d'une flèche sortante.

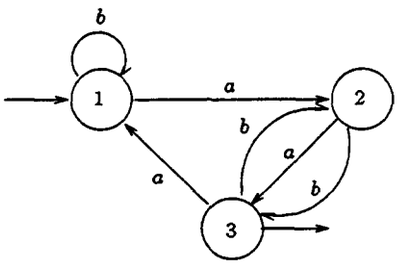

FIG. 5 – *L'automate* $\mathcal{S}$.

Un état p de $\mathcal{A}$ est dit *accessible* s'il existe au moins un mot u tel que $i \cdot u = p$. L'automate $\mathcal{A}$ est dit *accessible* si tous ses états le sont. L'automate $\mathcal{A}$ est dit *complet* si la fonction δ est bien définie sur tout $Q \times A$. On rappelle que tout automate accessible est équivalent à un automate complet et accessible. On sait par ailleurs que tout automate fini est équivalent à un automate fini déterministe et complet.

Problème

On se donne $\mathcal{A} = <Q, A, \delta, i, T>$ un automate déterministe et complet. Par définition, $\mathcal{A}$ est *régulier à droite* en ce sens que l'on a le résultat suivant que l'on ne demande pas d'établir :

$$\forall f, g \in A^* \quad i \cdot f = i \cdot g \implies [\forall v \in A^* \quad i \cdot fv = i \cdot gv] \tag{1}$$

On dira que $\mathcal{A}$ est *régulier à gauche* si

$$\forall f, g \in A^* \quad i \cdot f = i \cdot g \implies [\forall u \in A^* \quad i \cdot uf = i \cdot ug] \tag{2}$$

1. Vérifier que l'automate $\mathcal{S}$ représenté à la figure 5 *n'est pas* régulier à gauche.

2. On veut montrer que tout automate déterministe et complet est équivalent à un automate déterministe régulier à gauche. On va construire un automate dont l'ensemble des états est un sous-ensemble de $\mathcal{F}(Q)$, l'ensemble de toutes les applications de Q dans lui-même. Dans ce but, on procède aux identifications suivantes. Soit n le cardinal de Q, on identifie Q avec $[1, n] = \{1, 2, \ldots, n\}$ en prenant la convention que i est identifié à l'entier 1. Chaque élément r de $\mathcal{F}(Q)$ est identifié au vecteur $(r(1), r(2), \ldots, r(n))$ dont la k-ième coordonnée est l'image par r de l'élément $k \in [1, n]$. L'application identité, qu'on note j_Q, est donc représentée par le vecteur $(1, 2, \ldots, n)$. Par exemple, la permutation circulaire qui envoie 1 sur 2, 2 sur 3, *etc.*, $n-1$ sur n et n sur 1 est représentée par le vecteur $(2, 3, \ldots, n, 1)$.

On définit l'automate $\mathcal{A}_\mathcal{G} = <\mathcal{F}(Q), A, \eta, j_Q, U>$. La fonction de transition η est définie par: $\forall r \in \mathcal{F}(Q)$, $\forall a \in A, \eta(r,a) = r'$ avec $\forall k \in [1,n], r'(k) = \delta(r(k),a)$. L'état initial de $\mathcal{A}_\mathcal{G}$ est l'application identité, et l'ensemble U des états terminaux de $\mathcal{A}_\mathcal{G}$ est l'ensemble des applications qui envoient 1, l'état initial de $\mathcal{A}$, dans T , i.e. $U = \{r \in \mathcal{F}(Q) \mid r(1) \in T\}$. En fait, on s'intéresse seulement à la partie accessible de $\mathcal{A}_\mathcal{G}$ et on la note encore $\mathcal{A}_\mathcal{G}$.

Compléter la figure 6 pour donner la représentation graphique de $\mathcal{S}_\mathcal{G}$ correspondant à l'automate $\mathcal{S}$ de la figure 5. Il manque 9 transitions et l'indication du ou des états terminaux.

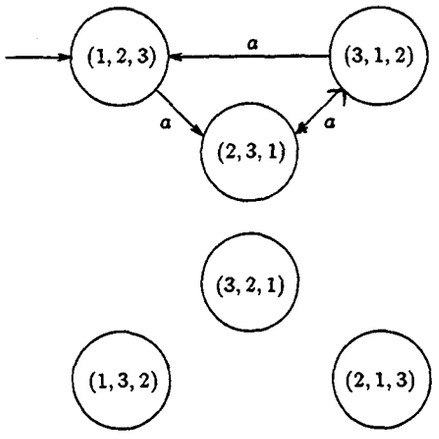

FIG. 6 – *L'automate $\mathcal{S}_\mathcal{G}$, à compléter.*

3. Vérifier que $\mathcal{A}_\mathcal{G}$ est un automate déterministe.
4. Montrer que $\forall r \in \mathcal{F}(Q)$, $\forall f \in A^*$, $\forall k \in [1,n]$, $(\eta(r,f))(k) = \delta(r(k),f)$.
5. Vérifier que $\mathcal{A}_\mathcal{G}$ est équivalent à $\mathcal{A}$.
6. Vérifier que $\mathcal{A}_\mathcal{G}$ est régulier à gauche.
7. Pour tout langage K de A^*, notons $\mathcal{R}_2(K) = \{f \in A^* \mid ff \in K\}$, l'ensemble des mots dont le carré est dans K. Soit $\mathcal{A} = <Q, A, \delta, i, T>$ un automate déterministe, on note $V_2 = \{p \in Q \mid \exists f \in A^* \text{ tel que } i \cdot f = p \text{ et } i \cdot ff \in T\}$ et $\mathcal{A}_{\mathcal{R}_2} = <Q, A, \delta, i, V_2>$ l'automate obtenu à partir de $\mathcal{A}$ en prenant V_2 comme ensemble d'états terminaux.

   Montrer que si $\mathcal{A}$ est régulier à gauche alors on a $L(\mathcal{A}_{\mathcal{R}_2}) = \mathcal{R}_2(L(\mathcal{A}))$.
8. En prenant $\mathcal{B} = \mathcal{S}_\mathcal{G}$, donner l'ensemble des états terminaux de $\mathcal{B}_{\mathcal{R}_2}$. On admettra, sans démonstration, qu'il suffit de considérer les mots dont le carré est de longueur inférieure ou égale à 6 pour calculer le V_2 correspondant.
9. Vérifier, avec l'automate $\mathcal{S}$ de la figure 5 que l'équation $L(\mathcal{A}_{\mathcal{R}_2}) = \mathcal{R}_2(L(\mathcal{A}))$ n'est pas toujours vérifiée pour un automate déterministe quelconque.
10. Conclusion et généralisation : soit k un entier positif quelconque, pour tout langage $K \subseteq A^*$ on note $\mathcal{R}_k(K) = \{f \in A^* \mid f^k \in K\}$. Montrer que si K est reconnu par un automate fini, $\mathcal{R}_k(K)$ est également reconnu par un automate fini.

FIN DU PROBLÈME SUR LES AUTOMATES

3 Problème de programmation en Pascal — 60 mn environ

UNIQUEMENT POUR LES CANDIDATS ET LES CANDIDATES CHOISISSANT LE LANGAGE PASCAL

L'objet de ce problème est la gestion des grands nombres encore appelés *big numbers*. En effet, la gestion des nombres entiers d'un langage de programmation est généralement liée à la spécificité du processeur de l'ordinateur. Avec un processeur 32 bits, un entier sera codé sur un mot mémoire de 32 bits et sera donc compris entre -2^{31} et $2^{31}-1$. Le but avec les *big numbers* est de s'affranchir des limitations induites par la capacité du processeur. L'énoncé propose deux méthodes pour mettre en œuvre les *big numbers*, entiers sans limitation de valeur.

N.B. Un entier du langage Pascal est codé sur un mot de 32 bits et donc sa valeur sera comprise entre -2^{31} et $2^{31}-1$ avec $2^{31} = 2147483648$.

Première partie.

Dans cette partie, nous choisissons de représenter un *big number* positif par la chaîne de caractères composée de ses chiffres et l'on désire réaliser une fonction d'addition des *big numbers* positifs. L'en-tête de cette fonction sera :

```
add_big_number(s1,s2 : string) : string ;
```

Elle s'utilisera de manière évidente :

```
add_big_number('11111222222333333','44555555888888')
   => '11155777778222221'
```

La méthode utilisée est la décomposition d'un *big number* en sous-chaînes de `N` chiffres et la réalisation de morceaux de somme afin d'obtenir le résultat. Pour les différents exemples qui suivent, nous avons choisi arbitrairement une valeur `N=6`. Nous appellerons *mot* une chaîne de caractères composée d'au plus `N` chiffres. Dans l'exemple plus haut, les deux chaînes arguments peuvent se décomposer de la façon suivante :

Première chaîne :	'11111'	'222222'	'333333'
Deuxième chaîne :	'44'	'555555'	'888888'

L'addition se fera de la droite vers la gauche. Le premier calcul est:

```
        333333
+       888888
-------------
     1  222221
```

On conserve de ce premier calcul la valeur de la retenue qui sera prise en compte pour le second calcul et on recommencera le mécanisme sur les deux mots suivants :

```
        222222
+       555555
+            1
-------------
        777778
```

On remarque qu'ici, il n'y a pas de retenue. On réalise enfin le troisième calcul :

```
    11111
+      44
-------
    11155
```

Le résultat final, `'11155777778222221'` est la concaténation des résultats partiels. Nous vous proposons de mettre en œuvre les différentes phases de cette méthode de calcul.

Le type `string`

On suppose l'existence du type `string` des chaînes de caractères. Une fonction peut avoir un résultat de type `string`. Les chaînes de caractères de type `string` peuvent être de longueur quelconque. Les fonctions de manipulation des chaînes de caractères sont décrites ci-dessous. On devra bien sûr les utiliser et n'utiliser que celles-ci en plus des fonctions et procédures que l'on écrira soi-même.

- une chaîne de caractères constante contenant les caractères `a,b,c,...` peut être écrite entre simples *quotes* : `'abc...'` ;
- `function concat(s1,s2 : string) : string ;`
  qui renvoie la concaténation de deux chaînes de caractères ;
- `function string_length(s : string) : integer ;`
  qui renvoie la longueur d'une chaîne ;
- `function char_to_string(c : char) : string`
  qui renvoie une chaîne contenant l'unique caractère `c` ;
- `function int_to_string(i : integer) : string ;`
  qui convertit un entier en une chaîne de caractères qui est sa représentation en base 10 ;
- `function string_to_int(s : string) : integer ;`
  qui convertit une chaîne de caractères représentant un entier en base 10 en l'entier ;
- `function nth_char(s : string; p : integer) : char ;`
  renvoie le caractère d'indice `p` dans la chaîne `s`. **Attention !** Le premier élément à partir de la gauche de la chaîne a pour indice 1. `p` doit être dans l'intervalle `1..string_length(s)`.

N.B. Dans les programmes qui sont demandés, on ne traitera pas les cas d'erreurs : on supposera que les arguments ont toujours des valeurs admissibles.

1. Pour réaliser l'addition de deux *big numbers*, déterminer la plus grande valeur possible pour `N`. Quel est l'intérêt de choisir cette valeur ?
2. Écrire la fonction `sub_string(s : string; lg,d : integer) : string` qui renvoie une chaîne de longueur `lg` contenant les caractères se trouvant entre l'indice `d` et l'indice `d+lg-1`.

3. Pour faciliter les calculs ultérieurs, tous les mots seront de même taille, c'est à dire composés de `N` chiffres. Écrire la fonction `canonize(s : string ; l : integer) : string` qui prend en premier argument une chaîne de longueur inférieure ou égale à son deuxième argument et ajoute un nombre suffisant de caractères 0 en tête de la chaîne pour que celle-ci ait exactement le deuxième argument comme longueur. Exemple :
```
canonize('1234',6)
  => '001234'
```
4. Écrire la fonction `words_count(s : string ; lg : integer) : integer` qui à partir d'une chaîne donnée en premier argument et d'un entier `lg` donné en deuxième argument détermine le nombre de mots de longueur `lg` qui la composent sachant qu'un mot, le premier, peut comporter entre 1 et `lg` caractères. Exemple :
```
words_count('11111222222333333',6)
  => 3
```
5. Écrire la fonction `aux_add_big_number(s1,s2 : string) : string` qui réalise l'addition de deux *big numbers* positifs codés sous la forme de leurs chaînes de caractères en supposant qu'ils sont constitués du même nombre de mots de longueur exactement `N`.
6. Écrire la fonction `add_big_number(s1,s2 : string) : string` qui réalise l'addition de deux *big numbers* positifs de longueurs quelconques.

Deuxième partie

7. On désire à présent utiliser le type `big_number` défini ci-dessous :
```
type
   big_number = ^element ;
   element = record
               mot     : integer    ;
               suivant : big_number ;
             end ;
```
Un *big number* est représenté par une liste chaînée d'`element`. Chaque élément contient la valeur numérique d'un mot de longueur `N`. Il contient également un pointeur sur l'`element` suivant s'il existe ou `nil` s'il n'existe pas. **Attention !** Dans cette représentation, les mots d'un *big number* sont listés du dernier vers le premier, c'est-à-dire de la droite vers la gauche du *big number*.

Écrire une fonction `add_new_big_number(n1,n2 : big_number) : big_number` réalisant l'addition de deux *big numbers* directement dans leur nouvelle représentation.

8. Écrire une fonction `big_number_to_string(n : big_number) : string` qui convertit un *big number* dans sa nouvelle représentation en une chaîne de caractères représentant le même *big number*.
9. Écrire une fonction `string_to_big_number(s : string) : big_number` qui convertit une chaîne de caractères représentant un *big number* positif en sa nouvelle représentation telle que décrite ci-dessus.
10. Réécrire alors la fonction `add_big_number` de la question 6 en utilisant très simplement les fonctions des questions 7, 8 et 9.

FIN DU PROBLÈME DE PROGRAMMATION EN PASCAL

4 Problème de programmation en Caml — 60 mn environ

UNIQUEMENT POUR LES CANDIDATS ET LES CANDIDATES CHOISISSANT LE LANGAGE CAML

L'objet de ce problème est la gestion des grands nombres encore appelés *big numbers*. En effet, la gestion des nombres entiers d'un langage de programmation est généralement liée à la spécificité du processeur de l'ordinateur. Avec un processeur 32 bits, un entier sera codé sur un mot mémoire de 32 bits et sera donc compris entre -2^{31} et $2^{31}-1$. Le but avec les *big numbers* est de s'affranchir des limitations induites par la capacité du processeur. L'énoncé propose deux méthodes pour mettre en œuvre les *big numbers*, entiers sans limitation de valeur.

N.B. Un entier du langage Caml est géré sur un mot de 31 bits et donc sa valeur sera comprise entre -2^{30} et $2^{30}-1$ avec $2^{30} = 1073741824$.

Première partie.

Dans cette partie, nous choisissons de représenter un *big number* positif par la chaîne de caractères composée de ses chiffres et l'on désire réaliser une fonction d'addition des *big numbers* positifs. La signature de cette fonction sera:

```
add_big_number : string -> string -> string
```

Elle s'utilisera de manière évidente:

```
# add_big_number "11111222222333333" "44555555888888" ;;
- : string = "11155777778222221"
```

La méthode utilisée est la décomposition d'un *big number* en sous-chaînes de `N` chiffres et la réalisation de morceaux de somme afin d'obtenir le résultat. Pour les différents exemples qui suivent, nous avons choisi arbitrairement une valeur `N=6`. Nous appellerons *mot* une chaîne de caractères composée d'au plus `N` chiffres. Dans l'exemple plus haut, les deux chaînes arguments peuvent se décomposer de la façon suivante:

Première chaîne:	"11111"	"222222"	"333333"
Deuxième chaîne:	"44"	"555555"	"888888"

L'addition se fera de la droite vers la gauche. Le premier calcul est:

```
      333333
+     888888
-------------
    1 222221
```

On conserve de ce premier calcul la valeur de la retenue qui sera prise en compte pour le second calcul et on recommencera le mécanisme sur les deux mots suivants:

```
      222222
+     555555
+          1
-------------
      777778
```

On remarque qu'ici, il n'y a pas de retenue. On réalise enfin le troisième calcul :

```
  11111
+    44
-------
  11155
```

Le résultat final, "11155777778222221" est la concaténation des résultats partiels. Nous vous proposons de mettre en œuvre les différentes phases de cette méthode de calcul. Vous pouvez utiliser les fonctions suivantes :

- `prefix ^ : string -> string -> string`
  qui renvoie la concaténation de deux chaînes de caractères ;
- `string_length : string -> int`
  qui renvoie la longueur d'une chaîne ;
- `string_of_char : char -> string`
  qui renvoie une chaîne contenant l'unique caractère donné en argument ;
- `string_of_int : int -> string`
  qui convertit un entier en une chaîne de caractères qui est sa représentation en base 10 ;
- `int_of_string : string -> int`
  qui convertit une chaîne de caractères représentant un entier en base 10 en l'entier ;
- `nth_char : string -> int -> char`
  qui renvoie le caractère se trouvant à la position déterminée par le second argument entier dans le premier argument. **Attention !** Le premier élément à partir de la gauche de la chaîne de caractères a pour indice `0`. Le deuxième argument doit être dans l'intervalle `0..(string_length s - 1)` où `s` est le premier argument.

N.B. Dans les programmes qui seront demandés, on ne traitera pas les cas d'erreurs : on supposera que les arguments ont toujours des valeurs admissibles.

1. Pour réaliser l'addition de deux *big numbers*, déterminer la plus grande valeur possible pour `N`. Quel est l'intérêt de choisir cette valeur ?
2. Écrire la fonction `sub_string : string -> int -> int -> string` telle que l'appel `(sub_string s lg d)` renvoie une chaîne de longueur `lg` contenant les caractères se trouvant entre l'indice `d` et l'indice `d+lg-1`.
3. Pour faciliter les calculs ultérieurs, tous les mots seront de même taille, c'est à dire composés de `N` chiffres. Écrire la fonction `canonize : string -> int -> string` qui prend en premier argument une chaîne de longueur inférieure ou égale à son deuxième argument et ajoute un nombre suffisant de caractères `0` en tête de la chaîne pour que celle-ci ait exactement le deuxième argument comme longueur. Exemple :

```
#canonize "1234" 6 ;;
- : string = "001234"
```

4. Écrire la fonction `words_count : string -> int -> int` qui à partir d'une chaîne donnée en premier argument et d'un entier `lg` donné en deuxième argument détermine le nombre de mots de longueur `lg` qui la compose sachant qu'un mot, le premier, peut comporter entre `1` et `lg` caractères . Exemple :

```
#words_count "11111222222333333" 6 ;;
- : int = 3
```

5. Écrire la fonction `aux_add_big_number : string -> string -> string` qui réalise l'addition de deux *big numbers* positifs codés sous la forme de leurs chaînes de caractères en supposant que ces deux *big numbers* sont constitués du même nombre de mots de longueur exactement `N`.

6. Écrire la fonction `add_big_number : string -> string -> string` qui réalise l'addition de deux *big numbers* positifs de longueurs quelconques.

Deuxième partie

7. On désire à présent utiliser le type `big_number` défini ci-dessous :

```
type element = {mutable mot : int} ;;
type big_number == element list ;;
```

Un *big number* est représenté par une liste d'`element`. Chaque élément contient la valeur numérique d'un mot de longueur `N`. **Attention !** Dans cette représentation, les mots d'un *big number* sont listés du dernier vers le premier, c'est-à-dire de la droite vers la gauche du *big number*.

Écrire une fonction `add_new_big_number : big_number -> big_number -> big_number` réalisant l'addition de deux *big numbers* directement dans leur nouvelle représentation.

8. Écrire une fonction `string_of_big_number : big_number -> string` qui convertit un *big number* dans sa nouvelle représentation en une chaîne de caractères représentant le même *big number*.

9. Écrire une fonction `big_number_of_string : string -> big_number` qui convertit une chaîne de caractères représentant un *big number* positif en sa nouvelle représentation telle que décrite ci-dessus.

10. Réécrire alors la fonction `add_big_number` de la question 6 en utilisant très simplement les fonctions des questions 7, 8 et 9.

FIN DU PROBLÈME DE PROGRAMMATION EN CAML

FIN DE L'ÉPREUVE