ÉPREUVE D'INFORMATIQUE

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

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 est amené(e) 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 vous 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 théorie des automates, n° 1 page 2, à résoudre en 60 minutes environ ;
- **Deuxième problème :** le problème de logique, n° 2 page 5, à résoudre en 30 minutes environ ;
- **Troisème problème :** un problème de programmation au choix parmi deux, à résoudre en 90 minutes environ. **Attention !** Pour le troisième problème, un seul choix est possible. Le candidat 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 6, pour les candidats et les candidates désirant choisir ce langage ;
  - le problème de programmation en langage Caml, le problème n° 4 page 9, pour les candidats et les candidates désirant choisir ce langage.

1 Problème sur les automates — 60 mn environ

La reconnaissance de motifs

L'objet du problème est de montrer que le modèle des automates finis conduit naturellement à la construction d'algorithmes efficaces pour la reconnaissance de motifs dans un texte. La reconnaissance de motifs est une fonction essentielle des logiciels de traitement de texte.

Rappels et notations

On appelle *alphabet* un ensemble fini A. Les éléments de A sont appelés des *lettres*. On note A^* l'ensemble des suites finies d'éléments de A qui sont appelées des *mots sur* A ou simplement des mots lorsqu'aucune ambiguïté n'est à craindre. Le mot vide est noté 1_{A^*}.

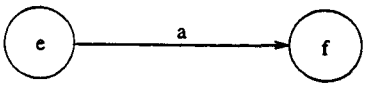
Soient $f, g \in A^*$, on note fg le mot obtenu en faisant suivre la suite des lettres de f par la suite des lettres de g, $fg \in A^*$. Soit $f \in A^*$, s'il existe $g, h \in A^*$ tels que $f = gh$ alors g est un *préfixe* de f et h est un *suffixe* de f. Si $h \neq f$, h est dit un *suffixe propre* de f et g est dit un *préfixe non-vide* de f. S'il existe $u, v, w \in A^*$ tels que $f = uvw$, v est un *facteur* de f. Enfin, on note $|f|$ la *longueur* de f, c'est-à-dire le nombre de lettres de l'alphabet qui le composent.

Un *automate* est une structure $< Q, A, E, I, T >$ telle que Q est un ensemble fini d'éléments appelés *états*, A est un alphabet, $I \subseteq Q$, les éléments de I sont appelés *états initiaux*, $T \subseteq Q$, les éléments de T sont appelés *états terminaux*, $E \subseteq Q \times A \times Q$, les éléments de E sont appelés des *transitions*.

Un *automate* est représenté graphiquement par un graphe orienté de la manière suivante : chaque état e est représenté par un noeud du graphe, chaque transition (e, a, f) est représentée par une flèche étiquetée par le mot a et qui va de l'état e vers l'état f. Pour marquer qu'un état est initial on ajoutera une flèche entrante sans origine. Pour marquer qu'un état est terminal on ajoutera une flèche sortante sans but. La figure 1 illustre ce mode de représentation graphique.



Un noeud correspondant à l'état e.



Une transition (e, a, f).



Un état initial e et un état final f.

FIG. 1 – *Représentation graphique d'un automate.*

Si pour chaque état $e \in Q$ et pour chaque lettre $a \in A$, il existe *au plus* un état $f \in Q$ tel que (e, a, f) soit une transition de l'automate, on dit que l'automate est *déterministe*. Dans ce cas, si $(e, a, f) \in E$ on choisit alors de noter f par $e \cdot a$. On n'attachera pas d'importance au fait que la notation $e \cdot a$ n'est pas définie pour tous les couples $(e, a) \in Q \times A$.

N.B. Dans tout le problème, l'alphabet A est fixé : $A = \{a, b\}$.

1. On considère l'automate $\mathcal{A}_{aba}$ décrit graphiquement par la figure 2.

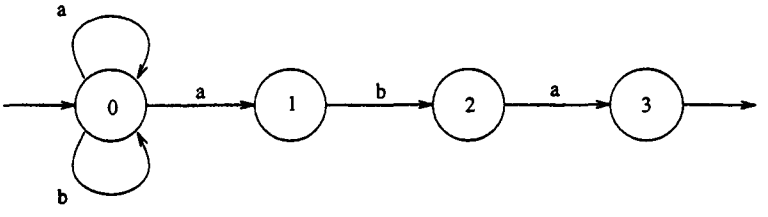


FIG. 2 – *L'automate* $\mathcal{A}_{aba}$.

   (a) Décrivez le langage reconnu par l'automate $\mathcal{A}_{aba}$.
   (b) Déterminisez l'automate $\mathcal{A}_{aba}$. Donnez la table de transition de l'automate déterminisé ainsi que sa représentation graphique.

2. On considère l'automate $\mathcal{A}_{abaab}$ décrit graphiquement par la figure 3.

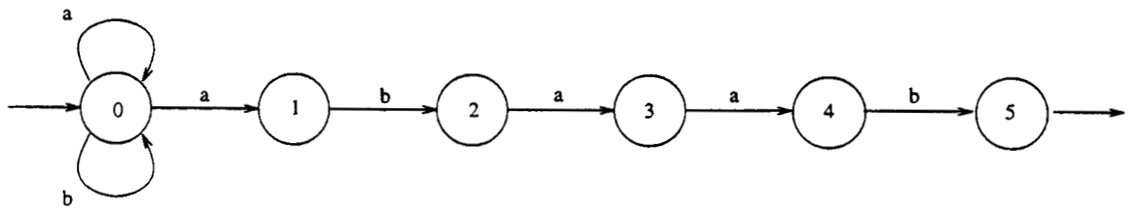


FIG. 3 – *L'automate* $\mathcal{A}_{abaab}$.

(a) Décrivez le langage reconnu par l'automate $\mathcal{A}_{abaab}$.

(b) Déterminisez l'automate $\mathcal{A}_{abaab}$. Donnez uniquement la table de transition de l'automate déterminisé.

Dans la suite du problème, on considère un mot $f = a_1 \cdots a_n \in A^*$ et l'automate $\mathcal{A}_f$ décrit graphiquement par la figure 4 page suivante.

On note P l'ensemble des *préfixes* du mot f :

$$P = \{1_{A^*},\ a_1,\ a_1a_2,\ \ldots,\ a_1 \cdots a_{n-1},\ a_1 \cdots a_n\}$$

L'ensemble des états de l'automate $\mathcal{A}_f$ peut être identifié avec l'ensemble des préfixes de f : l'état 0 est identifié avec 1_{A^*} et pour $i \in [1, n]$, l'état i est identifié avec le préfixe $a_1 \cdots a_i$. On note $\mathcal{D}_f$ le déterminisé de l'automate $\mathcal{A}_f$ obtenu par la méthode des sous-ensembles. L'ensemble des états de $\mathcal{D}_f$ est un sous-ensemble de l'ensemble des parties de P.

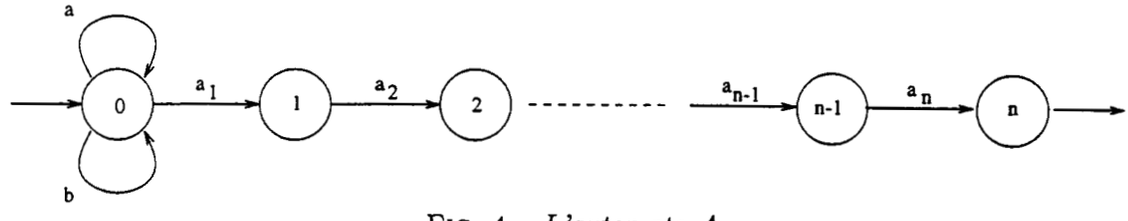


FIG. 4 – *L'automate* $\mathcal{A}_f$.

3. Soit $w \in A^*$ un mot quelconque.

   (a) Montrez que l'état de $\mathcal{D}_f$ atteint à partir de l'état initial par la lecture de w est le sous-ensemble de P constitué des suffixes de w.

   (b) Montrez que cet ensemble est entièrement déterminé par son élément le plus long. Dans la suite du problème, on identifiera cet ensemble avec son élément le plus long.

   (c) Déduisez de ce qui précède le nombre d'états de $\mathcal{D}_f$.

4. Soit α la fonction de $P - \{1_{A^*}\}$ dans P qui associe à chaque préfixe non-vide p le plus long suffixe propre de p qui est dans P. La fonction α n'est pas définie pour le mot vide 1_{A^*}.

   (a) Complétez la table 1 en donnant les valeurs de α dans le cas où f est le mot *abaab*.

p	a	ab	aba	$abaa$	$abaab$
$\alpha(p)$					

TAB. 1 – *La fonction* α *pour* $f = abaab$.

(b) Montrez que les transitions de $\mathcal{D}_f$ sont *bien définies* récursivement par :

$$\forall x \in A, \begin{cases} 1_{A^*} \cdot x &= x & \text{si } x \in P \\ &= 1_{A^*} & \text{sinon} \end{cases}$$

et :

$$\forall x \in A, \forall p \in P - \{1_{A^*}\}, \begin{cases} p \cdot x &= px & \text{si } px \in P \\ &= \alpha(p) \cdot x & \text{sinon} \end{cases}$$

5. Montrez que la fonction α elle-même peut être calculée récursivement par :

$$\forall x \in A, \alpha(x) = 1_{A^*}$$

et :

$$\forall x \in A, \forall p \in P - \{1_{A^*}\}, \begin{cases} \alpha(px) &= \alpha(p)x & \text{si } \alpha(p)x \in P \\ &= \alpha(\alpha(p)x) & \text{sinon} \end{cases}$$

6. Déduisez de ce qui précède un algorithme qui reconnaît toutes les occurrences d'un facteur f dans un mot t donné avec une complexité en $O(|f| + |t|)$.

FIN DU PROBLÈME SUR LES AUTOMATES

2 Problème de logique — 30 mn environ

Le connecteur de Sheffer

Le connecteur de Sheffer, noté |, est un connecteur binaire de la logique propositionnelle. Ce connecteur est défini par :

$$F|G \equiv_{def} \overline{F} \vee \overline{G}$$

où F et G sont des formules de la logique propositionnelle, $\overline{F}$ et $\overline{G}$ sont les *négations* ou *complémentations* de ces formules et $\equiv_{def}$ est un symbole d'*égalité par définition.*

Notations. On utilisera les lettres minuscules $x, y, z, \ldots$ pour désigner les variables propositionnelles. $\vee$ est le symbole pour le connecteur de *disjonction.* $\wedge$ est le symbole pour le connecteur de *conjonction.* $\Rightarrow$ est le symbole pour le connecteur d'*implication.*

On considère la logique propositionnelle munie des connecteurs de négation, de conjonction, de disjonction, d'implication et du connecteur de Sheffer.

L'objet de l'exercice est de montrer que pour toute formule F de la logique propositionnelle, il existe une formule F^*, ne contenant que des variables propositionnelles et le connecteur de Sheffer, telle que F et F^* soient logiquement équivalentes.

1. Construisez la table de vérité d'une formule $x|y$.
2. Construisez la table de vérité de la formule $x|x$.
3. Montrez que les formules $\overline{x}$ et $x|x$ sont logiquement équivalentes.
4. Montrez que $x \vee y$ et $(x|x)|(y|y)$ sont des formules logiquement équivalentes.
5. En vous servant des questions précédentes, montrez qu'il existe une formule logiquement équivalente à $x \Rightarrow y$ ne contenant que les variables propositionnelles x et y et deux occurrences du connecteur de Sheffer.

6. De même, montrez qu'il existe une formule logiquement équivalente à $x \wedge y$ ne contenant que les variables propositionnelles x et y et des occurrences du connecteur de Sheffer.

7. Des questions précédentes, déduisez un algorithme permettant de transformer toute formule F de la logique propositionnelle en une formule F^* ne contenant que des occurrences du connecteur de Sheffer et les variables propositionnelles déjà présentes dans F.

8. Appliquez l'algorithme de la question 7 à la formule $x \wedge (y \vee z)$.

   Définition 1. Soit F une formule de la logique propositionnelle, on appelle *poids* de F et on note $\sigma(F)$ le nombre de connecteurs présents dans la formule.

   Définition 2. On appelle *formule équilibrée* une formule F sans connecteur de négation qui vérifie les propriétés suivantes :

   - si F est une variable propositionnelle, F est équilibrée par définition ;
   - si F peut s'écrire $G\ c\ H$ avec $c \in \{\vee, \wedge, \Rightarrow, |\}$, si G et H sont équilibrées et si $\sigma(G) = \sigma(H)$ alors F est équilibrée.

9. Démontrez que si F est une formule équilibrée alors $\sigma(F)$ est de la forme $2^k - 1$ avec k entier naturel. Que représente k ?

10. On note $\sigma^*(n)$ le poids au pire de la transformée F^* d'une formule F équilibrée de poids n. Calculez $\sigma^*(n)$.

FIN DU PROBLÈME DE LOGIQUE

3 Problème de programmation en Pascal — 90 mn environ

UNIQUEMENT POUR LES CANDIDATS ET LES CANDIDATES CHOISISSANT LE LANGAGE PASCAL

La gestion des macro-définitions

L'objet de ce problème est de poser les bases d'un gestionnaire de macro-définitions. Les macro-définitions sont utilisées dans la plupart des langages informatiques : langages de programmation, langages de traitement de texte, langages de commandes, etc.

Les macro-définitions

On appelle *macro-définition* l'association d'une chaîne de caractères éventuellement vide à un mot. Le mot est alors appelé un *symbole actif* et la chaîne de caractères est sa *définition.* Les mots sont construits à l'aide des lettres de l'alphabet $\{a, \ldots, z,\ A, \ldots, Z,\ 0, \ldots, 9\}$. Les caractères espace et virgule, dollar et les parenthèses sont appelés des *séparateurs* Les chaînes de caractères sont écrites entourées par des guillemets (") permettant de les délimiter. Elles ne peuvent contenir que les caractères de l'alphabet et des séparateurs. La table 2 page suivante donne des exemples de macro-définitions : chaque ligne de la table contient un symbole actif et sa définition. Lorsque la définition contient des sous-chaînes de la forme `$i` avec `i` entier strictement positif, ces sous-chaînes désignent des arguments. Nous en verrons l'usage plus loin.

Un programme appelé *macro-expanseur* traite une chaîne de caractères et l'*expanse* en fonction des macro-définitions. On suppose que les chaînes traitées par le macro-expanseur commencent par un espace et se terminent par un espace. Une *macro-expression* dans la chaîne de caractères est composée d'un symbole actif qui doit être précédé d'un séparateur, puis d'une parenthèse ouvrante, d'un texte éventuellement vide et d'une parenthèse fermante ; dans le texte, les virgules qui apparaissent séparent des sous-chaînes qui sont appelées des *arguments* et qui sont numérotées consécutivement à partir de 1.

Exemple. Dans " `abcd Cat(abc de,uvw,nbc)` ", `"Cat(abc de,uvw,nbc)"` est une macro-expression dont les trois arguments sont les chaînes `"abc de"`, `"uvw"` et `"nbc"`.

Symbole actif	Définition
`Info`	`"Informatique"`
`Permute`	`"$2$1"`
`Echange`	`"$2,$1"`
`Cat`	`"$1$2"`
`Commentaire`	`""`
`x`	`"xx()"`
`xx`	`"xxx()"`
`y`	`"y()"`

TAB. 2 – *Exemples de macro-définitions.*

Le macro-expanseur prend une chaîne de caractères et expanse les macro-expressions qu'elle contient de la manière suivante : la macro-expression est remplacée par la définition associée au symbole actif puis, dans cette définition, chaque sous-chaîne `$i` est remplacée par le `i`-ième argument du symbole actif. Lorsque le `i`-ième argument n'existe pas, `$i` est remplacée par une chaîne vide. Ce processus est appelé *macro-expansion*.

Exemple. En prenant en compte les macro-définitions de la table 2, la macro-expansion de la chaîne " `Permute(Pre,Sec)` " fournit la chaîne " `SecPre` ".

N.B. Dans tout le problème, on considère les macro-définitions données dans la table 2.

1. Donnez les macro-expansions des chaînes de caractères suivantes :
   (a) la chaîne " `Info() Info1()` " ;
   (b) la chaîne " `Cat(1,2)` " ;
   (c) la chaîne " `Echange(1,2)` " ;
   (d) la chaîne " `a Commentaire(Hello,World) b` ".
2. Quels sont les deux résultats envisageables de la macro-expansion de la chaîne de caractères " `Permute(Info(),Epreuve)` " ?
3. Quand une chaîne contient des macro-expressions, celles-ci sont ordonnées de gauche à droite en fonction de la position de leur premier caractère.
   Exemple. Dans la chaîne " `Permute(Info(),Epreuve)` ", la macro-expression `"Info()"` est à droite de la macro-expression `"Permute(Info(),Epreuve)"` puisque le `I` de `Info` est à droite du `P` de `Permute`.

On décide que le macro-expanseur traite une chaîne suivant l'algorithme suivant : tant que la chaîne contient une macro-expression, choisir la macro-expression la plus à droite dans la chaîne et l'expanser suivant les règles.

Quel est alors le résultat de la macro-expansion de " `Permute(Info,Epreuve)` "?

4. Donnez les macro-expansions des chaînes de caractères suivantes, justifiez votre résultat par la suite des résultats intermédiaires :

   (a) la chaîne " `Echange(Echange(1,2))` ";

   (b) la chaîne " `Cat(In,fo())` ";

   (c) la chaîne " `x()` ";

   (d) la chaîne " `y()` ";

   (e) la chaîne " `x() y()` ";

   (f) la chaîne " `Cat(C,Cat(Echange(t,a)))(1,2)` ".

   N.B. Dans la suite du problème, on attachera une grande importance au traitement des erreurs dans les programmes qui vous sont demandés. En cas de détection d'erreur, un diagnostic clair doit être donné à l'utilisateur puis le programme doit se terminer.

   Les macro-définitions sont mémorisées dans une *table des symboles*. C'est un tableau dont les entrées contiennent un symbole actif et sa définition. C'est une variable globale du programme. Voici sa déclaration :

```
const
   tableMax = 128 ;

type
   macro_definition =          (* une entree du tableau *)
      array[1..2] of string ; (* symbole et definition *)
   table =                                  (* la table *)
      array[1..tableMax] of macro_definition ;

var
   TableDesSymboles : table ;
   TailleTable      : integer ;
```

   `TableDesSymboles` est la table des symboles et la variable `TailleTable` contient le nombre de macro-définitions mémorisées dans la table. Si la valeur de `TailleTable` est strictement positive, les macro-définitions sont mémorisées entre les indices `1` et `TailleTable`.

5. Donnez le code d'une fonction

```
estDansTable(s : string) : boolean ;
```

   dont le résultat est `true` si `s` est un symbole actif, `false` sinon. Quelle est sa complexité au pire en fonction du nombre d'entrées dans la table?

6. Donnez le code d'une fonction

```
defDansTable(s : string) : string ;
```

dont le résultat est la définition du symbole actif `s`. Quelle est sa complexité en fonction du nombre d'entrées dans la table?

On se propose d'améliorer *strictement* la mémorisation des macro-définitions et la complexité des précédentes opérations sur la table des symboles. Pour cela on se donne une fonction d'ordre sur les chaînes de caractères dont l'en-tête est :

```
function avant(chaine1,chaine2 : string) : boolean ;
```

On suppose que les entrées dans la table des symboles sont triées en fonction de leurs symboles actifs par cette fonction d'ordre.

7. Donnez le nouveau code de la fonction `defDansTable`. Quelle est sa complexité au pire en fonction du nombre d'entrées contenues dans la table?

8. Donnez le nouveau code de la fonction `estDansTable`. Quelle est sa complexité au pire en fonction du nombre d'entrées contenues dans la table?

FIN DU PROBLÈME DE PROGRAMMATION EN PASCAL

4 Problème de programmation en Caml — 90 mn environ

UNIQUEMENT POUR LES CANDIDATS ET LES CANDIDATES CHOISISSANT LE LANGAGE CAML

La gestion des macro-définitions

L'objet de ce problème est de poser les bases d'un gestionnaire de macro-définitions. Les macro-définitions sont utilisées dans la plupart des langages informatiques : langages de programmation, langages de traitement de texte, langages de commandes, etc.

Les macro-définitions

On appelle *macro-définition* l'association d'une chaîne de caractères éventuellement vide à un mot. Le mot est alors appelé un *symbole actif* et la chaîne de caractères est sa *définition.* Les mots sont construits à l'aide des lettres de l'alphabet $\{a, \ldots, z,\ A, \ldots, Z,\ 0, \ldots, 9\}$. Les caractères espace et virgule, dollar et les parenthèses sont appelés des *séparateurs.* Les chaînes de caractères sont écrites entourées par des guillemets (") permettant de les délimiter. Elles ne peuvent contenir que les caractères de l'alphabet et des séparateurs. La table 3 page suivante donne des exemples de macro-définitions : chaque ligne de la table contient un symbole actif et sa définition. Lorsque la définition contient des sous-chaînes de la forme `$i` avec `i` entier

Symbole actif	Définition
Info	"Informatique"
Permute	"$2$1"
Echange	"$2,$1"
Cat	"$1$2"
Commentaire	""
x	"xx()"
xx	"xxx()"
y	"y()"

TAB. 3 – *Exemples de macro-définitions.*

strictement positif, ces sous-chaînes désignent des arguments. Nous en verrons l'usage plus loin.

Un programme appelé *macro-expanseur* traite une chaîne de caractères et l'*expanse* en fonction des macro-définitions. On suppose que les chaînes traitées par le macro-expanseur commencent par un espace et se terminent par un espace. Une *macro-expression* dans la chaîne de caractères est composée d'un symbole actif qui doit être précédé d'un séparateur, puis d'une parenthèse ouvrante, d'un texte éventuellement vide et d'une parenthèse fermante ; dans le texte, les virgules qui apparaissent séparent des sous-chaînes qui sont appelées des *arguments* et qui sont numérotées consécutivement à partir de 1.

Exemple. Dans " abcd Cat(abc de,uvw,nbc) ", "Cat(abc de,uvw,nbc)" est une macro-expression dont les trois arguments sont les chaînes "abc de", "uvw" et "nbc".

Le macro-expanseur prend une chaîne de caractères et expanse les macro-expressions qu'elle contient de la manière suivante : la macro-expression est remplacée par la définition associée au symbole actif puis, dans cette définition, chaque sous-chaîne $i est remplacée par le i-ième argument du symbole actif. Lorsque le i-ième argument n'existe pas, $i est remplacée par une chaîne vide. Ce processus est appelé *macro-expansion.*

Exemple. En prenant en compte les macro-définitions de la table 3, la macro-expansion de la chaîne " Permute(Pre,Sec) " fournit la chaîne " SecPre ".

Début du problème

N.B. Dans tout le problème, on considère les macro-définitions données dans la table 3.

1. Donnez les macro-expansions des chaînes de caractères suivantes :
   (a) la chaîne " Info() Info1() " ;
   (b) la chaîne " Cat(1,2) " ;
   (c) la chaîne " Echange(1,2) " ;
   (d) la chaîne " a Commentaire(Hello,World) b ".
2. Quels sont les deux résultats envisageables de la macro-expansion de la chaîne de caractères " Permute(Info(),Epreuve) " ?

3. Quand une chaîne contient des macro-expressions, celles-ci sont ordonnées de gauche à droite en fonction de la position de leur premier caractère.

   Exemple. Dans la chaîne " `Permute(Info(),Epreuve)` ", la macro-expression "`Info()`" est à droite de la macro-expression "`Permute(Info(),Epreuve)`" puisque le `I` de `Info` est à droite du `P` de `Permute`.

   On décide que le macro-expanseur traite une chaîne suivant l'algorithme suivant : tant que la chaîne contient une macro-expression, choisir la macro-expression la plus à droite dans la chaîne et l'expanser suivant les règles.

   Quel est alors le résultat de la macro-expansion de " `Permute(Info,Epreuve)` "?

4. Donnez les macro-expansions des chaînes de caractères suivantes, justifiez votre résultat par la suite des résultats intermédiaires :

   (a) la chaîne " `Echange(Echange(1,2))` ";

   (b) la chaîne " `Cat(In,fo())` ";

   (c) la chaîne " `x()` ";

   (d) la chaîne " `y()` ";

   (e) la chaîne " `x() y()` ";

   (f) la chaîne " `Cat(C,Cat(Echange(t,a)))(1,2)` ".

   N.B. Dans la suite du problème, on attachera une grande importance au traitement des erreurs dans les programmes qui vous sont demandés. En cas de détection d'erreur, un diagnostic clair doit être donné à l'utilisateur puis le programme doit se terminer.

   Les macro-définitions sont mémorisées dans une *table des symboles*. C'est un tableau dont les entrées contiennent un symbole actif et sa définition. C'est une variable globale du programme. Voici sa déclaration :

```
let  tableMax = 128 ;;

type macro_definition = {          (* une entree du tableau *)
     Symbole : string ;                 (* le symbole actif *)
     Definition : string                   (* sa definition *)
} ;;

let  EntreeVide = { Symbole = "" ; Definition = "" } ;;

type 'a table_des_symboles = {
              Table : 'a vect ;
     mutable TailleTable : int
} ;;

let  TableDesSymboles = {                          (* la table *)
     Table = make_vect tableMax EntreeVide ;
     TailleTable = 0
} ;;
```

Dans `TableDesSymboles`, le champ `Table` est la table des symboles et le champ `TailleTable` contient le nombre de macro-définitions mémorisées dans la table. Si la valeur de `TailleTable` est strictement positive, les macro-définitions sont mémorisées entre les indices `0` et `TailleTable-1`.

5. Quel est le type de `TableDesSymboles`?

6. Donnez le code d'une fonction

```
estDansTable : string -> bool
```

dont le résultat est `true` si son argument est un symbole actif, `false` sinon. Quelle est sa complexité au pire en fonction du nombre d'entrées dans la table?

7. Donnez le code d'une fonction

```
defDansTable : string -> string
```

dont le résultat est la définition du symbole actif donné en argument. Quelle est sa complexité en fonction du nombre d'entrées dans la table?

On se propose d'améliorer *strictement* la mémorisation des macro-définitions et la complexité des précédentes opérations sur la table des symboles. Pour cela on se donne une fonction d'ordre sur les chaînes de caractères dont le type est :

```
avant : 'a -> 'a -> bool
```

On suppose que les entrées dans la table des symboles sont triées en fonction de leurs symboles actifs par cette fonction d'ordre.

8. Donnez le nouveau code de la fonction `defDansTable`. Quelle est sa complexité au pire en fonction du nombre d'entrées contenues dans la table?

9. Donnez le nouveau code de la fonction `estDansTable`. Quelle est sa complexité au pire en fonction du nombre d'entrées contenues dans la table?

FIN DU PROBLÈME DE PROGRAMMATION EN CAML

FIN DE L'ÉPREUVE

www.tunisie-etudes.info

Ce document a été téléchargé depuis
www.tunisie-etudes.info

Des documents gratuits, devoirs, examens, cours, exercices, corrigés... Ainsi que toute une rubrique pour vous aider à trouver un emploi sans oublier les avis de concours en direct

Notre page Twitter :
http://www.twitter.com/TunisieEtudes
Notre page FaceBook :
http://www.facebook.com/TunisieEtudes



Merci d'avoir choisi www.tunisie-etudes.info
Bonne lecture et bon travail