

Algorithmique et Base de la programmation

Durée : 1h30

Documents autorisés : **AUCUN**

Remarques :

- Veuillez lire attentivement les questions avant de répondre.
- Le barème donné est un barème indicatif qui pourra évoluer lors de la correction.
- Rendez une copie propre.
- Dans les exercices qui suivent vous pouvez utiliser les fonctions suivantes :
 - **fonction** longueur (uneChaine : **Chaîne de caracteres**) : **Naturel**
 ...avec $longueur("") = 0$
 - **fonction** iemeCaractere (uneChaine : **Chaîne de caracteres**, iemePlace : **Naturel**) : **Caractere**
 |precondition(s) $0 < iemePlace \leq longueur(uneChaine)$

1 Distance de Hamming

1.1 Distance de Hamming entre deux mots

La distance de Hamming entre deux mots (chaîne de caractères) de même longueur est égale au nombre de lettres, à la même position, qui diffère. Par exemple la distance de Hamming entre “rose” et “ruse” est de 1, alors que la distance de Hamming entre “110110” et “000101” est de 4.

Proposez le corps de la fonction **itérative** suivante qui permet de calculer la distance de Hamming de deux mots, non vides, que l’on sait de même longueur :

- **fonction** distanceHammingDeuxMots (mot1,mot2 : **Chaîne de caracteres**) : **Naturel**

|precondition(s) $mot1 \neq ""$ et $longueur(mot1) = longueur(mot2)$ ¹

1.2 Distance de Hamming d’un langage

Un langage est un ensemble de mots. La distance de Hamming d’un langage est égale au minimum des distances de Hamming entre deux mots de ce langage différents deux à deux.

Proposez le corps de la fonction **itérative** suivante qui permet de calculer la distance de Hamming d’un langage d’au moins 2 mots qui possède uniquement des mots de même longueur et tous différents deux à deux :

¹Cela signifie que vous n’avez pas à faire ce test dans votre algorithme

- **fonction** `distanceHammingLangage` (`leLangage` : **Tableau**[1..MAX] **de** **Chaine de caracteres** , `nbMots` : **Naturel**) : **Naturel**
|precondition(s) $nbMots \leq MAX$

2 Évaluation d’une expression booléenne

Soit une expression booléenne exprimée sous forme d’une chaîne de caractères, dont la syntaxe suit les caractéristiques suivantes :

- Notation préfixe
- Un caractère par symbole, avec :
V pour vrai
F pour faux
& pour l’opérateur **et**
| pour l’opérateur **ou**
! pour l’opérateur **non**

Par exemple la chaîne de caractères “!*&|V F V*” représente l’expression **non((vrai ou faux) et vrai)**.

Proposez le corps de la procédure **récursive** suivante qui évalue une expression booléenne que l’on sait bien formée :

- **procédure** `evaluer` (**E** `lExpression` : **Chaine de caracteres** , **E/S** `position` : **Naturel** , **S** `valeur` : **Booleen**)
avec les paramètres ayant les significations suivantes :
lExpression l’expression à évaluer,
position l’indice de début de l’expression (ou de la sous-expression) que l’on va évaluer (au premier appel, la valeur de la variable qui fait office de paramètre effectif, est de 1),
valeur la valeur de l’expression (ou de la sous-expression) que l’on est train d’évaluer.

3 Un peu de C

3.1 Des signatures de fonctions

Proposez les signatures C des fonctions des deux premiers exercices

3.2 De l’algorithme au C

En Travaux Dirigés, nous avons vu l’algorithme suivant :

procédure `calculerMinMax` (**E t** : **Tableau**[1..MAX] **d’Entier** ; `n` : **Naturel** , **S** `min,max` : **Entier**)

Déclaration `i` : **Naturel**

debut

```
min ← t[1]
max ← t[1]
pour i ← 2 à n faire
  si t[i] < min alors
    min ← t[i]
  finsi
  si t[i] > max alors
    max ← t[i]
  finsi
finpour
fin
```

Proposez la fonction C correspondante.