

Royaume du Maroc  
Agence Nationale de Réglementation des Télécommunications  
Institut National des Postes et Télécommunications

## Concours d'accès en 1<sup>ère</sup> année de l'Institut National des Postes et Télécommunications

Lundi 16 juillet 2012

### Epreuve d'algorithmique et programmation en Langage C

#### Recommandations aux candidats

- L'appréciation des copies tient compte de la rigueur algorithmique, de la présentation et de la clarté de la rédaction.
- Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie, propose une correction et poursuit l'épreuve en conséquence.
- L'usage de calculatrices électroniques n'est pas autorisé.

**Durée de l'épreuve: 1 heure**

---

# Problème: Validité des identificateurs d'un langage de programmation

## Contexte du problème

Tout langage de programmation permet l'utilisation de chaînes de caractères pour déclarer et identifier les constantes et les variables du programme.

Ces chaînes de caractères appelés **identificateurs** doivent respecter un ensemble de règles particulières pour chaque langage.

Le problème suivant s'intéresse à la vérification de la validité d'identificateurs d'un langage spécifique

## Règles de validité des identificateurs du langage à étudier

Soit un langage de programmation qui définit, comme suit, les règles de validité d'un identificateur:

- Un identificateur est une chaîne de caractères de longueur inférieure strictement à **80**
- Un identificateur ne peut contenir que des caractères alphabétiques minuscules ('a', ..., 'z') ou majuscules ('A', ..., 'Z') ou des chiffres ('0', '1', ..., '9').
- Un identificateur ne doit pas commencer par un **chiffre** ('0', '1', ..., '9').
- Un identificateur ne doit pas être un des mots clés suivants : **if**, **do**, **while**, **for**.

## Remarques

- Toutes les fonctions demandées seront écrites en **langage C**
- Les questions non traitées peuvent être admises pour aborder les questions ultérieures
- Toute fonction peut être décomposée, si nécessaire, en plusieurs fonctions
- Seules les fonctions suivantes définies dans la bibliothèque du langage **C** peuvent être appelées sans être définies :

- Fonctions déclarées dans le fichier **string.h**

**int strlen (const char\*)** retourne la longueur de la chaîne en paramètre

**char\* strcpy (char\*, const char\*)** : copie la 2ème chaîne dans la première

**int strcmp (const char\*, const char\*)** compare les 2 chaînes en paramètres et retourne **0** si elles sont identiques

- Fonction déclarée dans le fichier **stdlib.h**

**void free (void\*)** : libère l'espace dynamique alloué

- **Dans ce problème, il n'est pas demandé d'écrire la fonction main**

**Rappel** : Les valeurs décimales des codes **ASCII** des **caractères chiffres**, des **caractères alphabétiques majuscules** ou **minuscules** sont indiqués ci-dessous :

caractère	'0'	'1'	...	'9'
Code ASCII	48	49		57

'A'	'B'	....	'Z'
65	66		90

'a'	'b'	...	'z'
97	98		122

---

## 1- Vérification de la validité d'un identificateur

→ **Question 1** (5 points) : Ecrire une fonction de prototype **int valide (char id[ ])** qui retourne **1** si son paramètre **id** est un identificateur valide ou retourne **0** (zéro) sinon.

(Un identificateur est valide s'il respecte les règles du langage à étudier citées plus haut)

**Exemple :**

**Concours, INPT, Max10, Min0** sont des identificateurs valides.

**3xy, hy\*, if** ne sont pas des identificateurs valides.

## 2- Suppression des identificateurs non valides dans un tableau

→ **Question 2** (8 points): Soit **T** un tableau de **N** ( $0 < N$ ) chaînes de caractères dont les longueurs sont inférieures à **80**

Ecrire une fonction de prototype : **void supprimer(int N, char T[ ][80])** qui supprime toutes les chaînes qui ne correspondent pas à des identificateurs valides. Ces chaînes seront remplacées par des chaînes **vides** à la fin du tableau ( voir exemple)

**Exemple**

- Soit **N= 5** et **T**={"identif1", "4val", "variable", "if", "delta"}

Après l'appel de la fonction **supprimer(5, T)**, **T**={"identif1", "variable", "delta", "", ""}

## 3- Suppression des identificateurs redondants dans une liste chaînée

On suppose avoir **déjà crée une liste chaînée d'identificateurs valides** dans la mémoire dynamique. Cette liste chaînée est définie par le type **liste** déclaré ainsi :

```
typedef struct tliste
```

```
{ char identificateur[80]; // représente un identificateur
```

```
struct tliste *suiv; // représente l'adresse de l'élément suivant dans la liste
```

```
} liste;
```

→ **Question 3** (7 points): Ecrire une fonction de prototype :

**void supprimer\_redondants(liste \*adrDebut)** qui permet de supprimer tous les **identificateurs redondants** dans une liste chaînée de type **liste** définie plus haut.

Le paramètre (**adrDebut**) de cette fonction représente l'adresse du premier élément de cette liste, le dernier élément de cette liste a l'adresse **NULL** dans son champ **suiv**

Dans le cas où **n** ( $n > 1$ ) identificateurs sont identiques dans la liste chaînée, on ne laissera dans la liste que le premier et on supprimera les (**n-1**) autres (voir exemple)

---

### Exemple

-Représentation de la Liste chaînée avant la suppression des identificateurs redondants :

**adrDebut**

Valeur	→	nombre	→	nombre	→	total	→	Valeur	→	X10	→	total
suiv	→	suiv	→	suiv	→	suiv	→	suiv	→	suiv	→	NULL

-Représentation de la liste chaînée après l'appel de la fonction **supprimer\_redondants(adrDebut)**

**adrDebut**

Valeur	→	nombre	→	total	→	X10
suiv	→	suiv	→	suiv	→	NULL

/\*\*\*\*\* FIN DE L'EPREUVE \*\*\*\*\*/