ROYAUME DU MAROC
المملكة المغربية

Ministère de l'Enseignement Supérieur,
de la Recherche Scientifique et de la Formation des Cadres

Présidence du concours National commun
Ecole Nationale Supérieure de l'Informatique et d'Analyse des Systèmes

ENSIAS

CONCOURS NATIONAL COMMUN
d'admission dans les Établissements de Formation d'Ingénieurs
et Établissements Assimilés

Session 2012

ÉPREUVE D'INFORMATIQUE

Filière **MP /PSI/TSI**

Durée 2 heures

Cette épreuve comporte 8 pages au format A4, en plus de cette page de garde
L'usage de la calculatrice est (Interdit)

L'énoncé de cette épreuve, commune aux candidats des filières MP/ PSI/ TSI, comporte 8 pages.
L'usage de la calculatrice est interdit .

*Les candidats sont informés que la précision des raisonnements **algorithmiques** ainsi que le soin apporté à la rédaction et à la présentation des copies seront des éléments pris en compte dans la notation. Il convient en particulier de rappeler avec précision les références des questions abordées. Si, au cours de l'épreuve, un candidat repère ce qui peut lui sembler être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.*

Remarques générales :

- L'épreuve se compose de deux problèmes indépendants.
- Toutes les instructions et les fonctions demandées seront écrites en **langage C**.
- Les questions non traitées peuvent être admises pour aborder les questions ultérieures.

PROBLÈME I : REPRÉSENTATION DE GRANDS NOMBRES ENTIERS NATURELS

✧✧✧

Contexte du problème :

Pour tout langage de programmation, la représentation des nombres par des types prédéfinis est très limitée (en langage **C**, une variable entière **v** de type **long int** est bornée ainsi : **-2147483648 <= v <= 2147483647**).

Cependant, plusieurs problèmes (par exemple les systèmes GPS, la cryptographie ou encore les applications de gestion) ont besoin d'utiliser des nombres ayant des valeurs et des précisions qui dépassent largement les limites des types prédéfinis par les langages de programmation.

Pour toutes ces applications, les types de base ne conviennent plus et il faudra définir de nouvelles représentations pour ces nombres.

Le problème suivant s'inscrit dans ce contexte. Il s'intéresse à la représentation des nombres entiers positifs pouvant avoir des valeurs très grandes non spécifiées par les types **standard** du langage **C**.

A : Représentation par des chaînes de caractères

Dans cette partie, on représentera un nombre entier positif ou nul par une chaîne de caractères contenant ses chiffres.

Rappels

- Une chaîne de caractères en langage **C** est un tableau de caractères se terminant par le caractère spécial '\0'
- La longueur d'une chaîne de caractères est le nombre de caractères de la chaîne avant '\0'

Notations

- On dira qu'une Chaîne de caractères **S** est une **ChaineChiffres** si sa longueur est strictement positive et tout caractère de **S** est un caractère chiffre (les caractères chiffres sont '0','1','2','3','4','5','6','7','8','9')
- Soit **N** un nombre entier positif ou nul de **NC** chiffres (NC > 0), la valeur décimale (base 10) de **N** est $\mathbf{N} = C_{NC-1}C_{NC-2}\ldots C_i\ldots C_1C_0$, ($C_0$ chiffre des unités, C_1 chiffre des dizaines, ...), on dit que **N** est représenté par une **ChaineChiffres S**, si **S** contient les chiffres de **N** comme suit : $S = "C_{NC-1}C_{NC-2}\ldots C_i\ldots C_1C_0"$

Exemple :

Le nombre **N=45009876156430987** de 17 chiffres sera représenté par la **ChaineChiffres**

S= "45009876156430987" (S[0] = 4, S[1] = 5, S[2] = 0..., S[16] = 7)

Question 1 : Chaîne de chiffres

Soit **S** un une chaîne de caractères quelconque déjà déclarée et initialisée.

☞ Définir une fonction d'entête **int ChaineChiffres()** qui retourne 1 si **S** st une **ChaineChiffres** ou 0 (zéro) sinon.

Exemple :

- Si **S="78500120360007"** alors l'appel de la fonction **ChaineChiffres()** retourne 1
- Si **S="856942a1478"** alors l'appel de la fonction **ChaineChiffres()** retourne 0

Question 2 : Zéros non significatifs

On suppose que **S** est une **ChaineChiffres** déjà déclarée et définie.

☞écrire une fonction d'entête **void supprimer_zeros()** qui supprime les zéros à gauche de la chaîne **S** (les **zéros** non significatifs dans un nombre)

Exemple :

Si **S= "0009760004300"**, après l'appel de **supprimer_zeros(s)**, **S= "9760004300"**.

Question 3 : Somme de deux chaînes de chiffres

Il s'agit de faire la somme de deux nombres entiers positifs représentés par leurs **chaineChiffres.** Pour ce faire :

☞Écrire une fonction de prototype **void additionner(char S1[ ],char S2[ ],char SOM[ ])** ; qui place dans la chaîne **SOM** la somme des nombres représentés par les **ChaîneChiffres S1** et **S2**.

Rappel :

les valeurs décimales des codes ASCII des caractères chiffres sont indiqués dans le tableau qui

suit :

caractère	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
Code ASCI II	48	49	50	51	52	53	54	55	56	57

Exemple :
si **S1= "129782004977"** et **S2="7540229953"** alors après l'appel de la fonction **additionner(S1,S2,SOM)**, on aura **SOM="137322234930"**.

B : Représentation des nombres par des listes chaînées

Dans cette partie, on utilisera des listes chaînées pour représenter des nombres entiers positifs ou nuls.
Pour optimiser la représentation d'un nombre, on se propose de le décomposer en plusieurs parties. Chaque partie peut contenir 4 chiffres.
Soit **N** un nombre entier positif ou nul de **NC** chiffres. $\mathbf{N} = C_{NC-1}C_{NC-2}\dots C_i\dots C_1C_0$, ($C_0$ chiffre des unités, C_1 chiffre des dizaines, ...). Alors **N** sera décomposé ainsi :

$$N = \underbrace{\boxed{C_{NC-1}C_{NC-2}C_{NC-3}C_{NC-4}}}_{1^{ère}\,partie}\ \underbrace{\boxed{C_{NC-5}C_{NC-6}C_{NC-7}C_{NC-8}}}_{2^{ème}\,partie}\ \dots\ \underbrace{\boxed{\dots C_1C_0}}_{dernière\,partie}$$

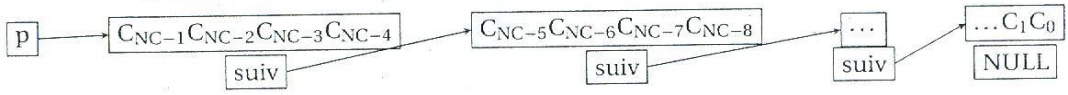
Exemple : Le nombre **N= 8002590300407896420003** peut être décomposé ainsi :

$$N = \underbrace{\boxed{8002}}_{1^{ère}\,partie}\ \underbrace{\boxed{5903}}_{2^{ème}\,partie}\ \underbrace{\boxed{0040}}_{3^{ème}\,partie}\ \underbrace{\boxed{7896}}_{4^{ème}\,partie}\ \underbrace{\boxed{4200}}_{5^{ème}\,partie}\ \underbrace{\boxed{03}}_{6^{ème}\,partie}$$

Pour représenter un nombre en le décomposant ainsi, on va utiliser une liste chaînée de type **ListeNombres** définie en langage **C** comme suit :

```
typedef struct Liste
        { short int partie;
        struct Liste *suiv; // l'adresse de l'élément suivant
        } ListeNombres;
```

Chaque partie du nombre est un élément de la liste chaînée contenant la valeur de la partie et l'adresse de la partie suivante. Le nombre $\mathbf{N} = C_{NC-1}C_{NC-2}\dots C_i\dots C_1C_0$ sera représenté par une liste de type **ListeNombres** comme suit :



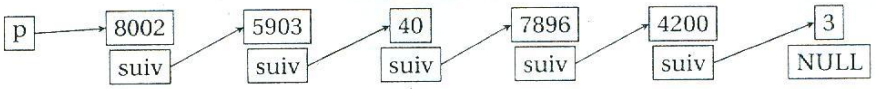
p (adresse du premier élément de la liste).

La valeur du champ partie étant déclarée de type **short int** , les **0 (zéros)** non significatifs (à gauche) n'apparaîtrons pas dans la valeur de la partie(voir exemple).

Exemple : Le nombre 8002590300407896420003 peut être décomposé ainsi :

N = | **8002** | **5903** | **40** | **7896** | **4200** | **3** |
|---|---|---|---|---|---|
| $1^{ère}$ *partie* | $2^{ème}$ *partie* | $3^{ème}$ *partie* | $4^{ème}$ *partie* | $5^{ème}$ *partie* | $6^{ème}$ *partie* |

Et sera représenté par une liste de type **ListeNombres**



Question 4 : Création d'une liste chaînée des parties d'un nombre

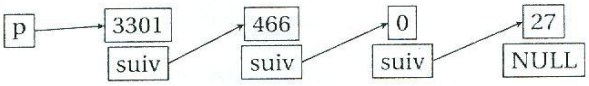
Soit un nombre entier positif ou nul défini par une **ChaineChiffres S** (voir définition de **ChaineChiffes** dans **la partie A** de ce problème). On se propose de le représenter par une liste chaînée de type **ListeNombres.** Pour ce faire

☞Écrire une fonction de prototype : **ListeNombres *regrouperChiffres(char *S)** ; qui permet de créer une nouvelle liste chaînée de type **ListeNombres** pour représenter le nombre défini par la **ChaineChiffres S** (en paramètre). Cette fonction retourne l'adresse du premier élément de cette liste crée.

Rappel : La fonction de prototype **void *malloc(int n) ;** définie dans le fichier de la bibliothèque du langage C **stdlib.h**, permet d'allouer un block de **n** octets dans la mémoire dynamique et retourne l'adresse mémoire de ce block alloué.

Exemple : Soit le nombre **330104660000027** représenté par une **ChaineChiffres S = 330104660000027**

L'appel de la fonction **regrouperChiffres(S)**, va retourner l'adresse du premier élément **p** de la liste crée ainsi :



PROBLÈME II : ALGORITHME DE CHIFFREMENT RC4

✧✧✧

Préambule

Le **"WIFI"** est une technologie de transmission radio qui permet d'échanger des données, sans se servir de fils. Cette technologie est utilisée pour connecter des ordinateurs à une borne afin d'accéder à un réseau local ou **Internet.**

La propagation des ondes radio à travers l'air étant naturellement non protégée, beaucoup

d'études et de recherches ont été réalisées pour sécuriser les données transmises dans les réseaux sans fils.

C'est dans cette perspective que le protocole **WEP (Wired Equivalent Privacy)** a été conçu pour assurer la sécurité des réseaux **WIFI.** Ce protocole implémente l'algorithme **RC4** objet de ce problème.

Principe général de l'algorithme RC4

RC4 "Rivest Cipher 4" est un algorithme de chiffrement (algorithme de cryptage) conçu par **Ronald Rivest en 1987.** Cet algorithme transforme à l'aide d'une clé secrète appelée **clé de chiffrement**, un texte clair en un texte incompréhensible (dit texte chiffré ou crypté) pour celui qui ne dispose pas de la clé de chiffrement.

Il est largement déployé dans les réseaux sans fils pour garantir leurs sécurités et dans plusieurs applications **E_COMMERCE** pour assurer la confidentialité des transactions **Internet.**

Description

RC4 permet de chiffrer un texte en générant une suite d'octets pseudo aléatoires. Cette suite produira le texte chiffré.
L'algorithme s'effectue en deux phases :

-**Une phase d'initialisation** visant à créer la table d'état (un tableau de 256 entiers) en utilisant la clé de chiffrement. Cet algorithme est appelé **Key Scheduling Algorithm, (KSA).**

- Une phase de génération d'octets pseudo aléatoires Cet algorithme est appelé **Pseudo Random Generator Algorithm (PRGA)**

Le problème suivant s'intéresse à l'implementation de l'algorithme **RC4 en langage C**

A : Chiffrement d'une chaîne de caractère par l'algorithme RC4

Dans cette partie, on se propose de chiffrer selon l'algorithme RC4, un texte sous forme d'une chaîne de caractères.

Déclarations globales préliminaires

Dans toutes les questions de la **partie A,** on suppose avoir déjà déclaré les constantes et variables globales suivantes :

- **L1** et **L2** deux constantes entières déjà définies (0<L1<256) et (0<L2<256)
- **Claire** une chaîne de **L1** caractères contenant le texte à chiffrer.
- **Clef** une chaîne de **L2** caractères contenant la clé de chiffrement.
- **Chiffree** une chaîne de **L1** caractères destinée à contenir le texte après son chiffrement
- **K** une chaîne de **256** caractères
- **S** un tableau de **256** entiers (la table d'état)

Phase 1 : Algorithme d'ordonnancement clé (Key Scheduling Algorithm (KSA))

L'algorithme (KSA) est décrit comme suit :

- Initialiser d'abord le tableau **S** (table d'état) avec l'identité
- Initialiser le tableau **K** avec la chaîne **Clef** (représentant la clé de chiffrement)
- Mélanger les éléments de **S** de la façon suivante :
  - déclarer deux indices entiers **i** et **j** et initialiser **j** à **0**
  - Pour tout **i** variant de **0 à 255**, faire les 2 instructions suivantes :
    Instruction 1 : **j = (j + S[i] + K[i]) modulo 256**
    Instruction 2 : **Permuter S[i]** et **S[j]**

Rappel : En langage C, l'opérateur modulo est représenté par le symbole %

Il s'agit dans cette **phase 1** de mettre en œuvre l'algorithme **(KSA)** en le décomposant en plusieurs fonctions.

Question 1 : Initialisation du tableau S avec l'identité

☞ Écrire la fonction de prototype **void identiteS( )** ; qui permet d'initialiser les éléments du tableau **S** de telle sorte que chaque élément **S[i]** (**0 <= i < 256**) soit initialisé par la valeur de l'indice **i** (**S[0] = 0, S[1] = 1, ... S[i] = i, ..., S[255] = 255**)

Question 2 : Initialisation du tableau K avec la clé

☞ Écrire la fonction de prototype **void initialiserK( )** ; qui permet d'initialiser la chaîne **K** avec les caractères de la chaîne **Clef** comme suit :

K[0]=Clef[0], K[1]=Clef[1],...,K[L2-1]=Clef[L2-1], K[L2]= Clef[0], K[L2+1]=Clef[1],..., K[255]=Clef[...]

Exemple : Soit la clé de longueur **3 (L2= 3)** suivante **Clef="ABC"** alors le tableau **K** sera initialisé ainsi : **K[0]='A', K[1]='B', K[2]='C',K[3]='A', K[4]='B',K[5]='C',..., K[255]='A'**

Question 3 : Permutation

☞ Écrire une fonction de prototype **void permuterS(int i,int j)** qui permet de permuter **S[i]** avec **S[j]** (on suppose **0<=i<256 et 0<=j<256)**

Question 4 : Algorithme KSA

☞ Écrire la fonction de prototype **void KSA ()** qui réalise l'algorithme **d'ordonnancement-clé (KSA)**, défini ci-dessus au début de ce problème.

Phase 2 : Pseudo Random Generator Algorithm (PRGA)

L'algorithme **(PRGA)** est décrit comme suit :

- Déclarer 3 entiers **i , j** et **a** puis initialiser **i** à **0** et initialiser **j** à **0**
- Déclarer un entier de nom **octet** (**octet** contiendra l'octet généré)
- Pour a variant de **0 à (L1-1)**, faire les instructions suivantes
  - **i= ((i+1) modulo 256)**
  - **j=((j +Claire[i]) modulo 256)**
  - Permuter **S[i]** et **S[j]**
  - **octet = S[(S[i]+S[j]) modulo 256]**
  - **Chiffree[a]= (Clef[a]) OU EXCLUSIF (octet)**

Pour réaliser cet algorithme, on se propose de le découper en plusieurs fonctions.

Question 5 : : Décomposition binaire

☞ Écrire une fonction de prototype **void decomposer(int bit[ ], int nombre)** ; qui met dans le tableau **bit** (1^{er} paramètre), la représentation binaire sur un octet (8 bits) de l'entier **nombre** ($2^{ème}$ paramètre) . On suppose **(0<=nombre<256)**
Dans cette décomposition, **bit[0]** contiendra le bit de poids faible(voir exemple)

Exemple :

Après l'appel de la fonction **decomposer(bit,11)**, on aura
bit([0]=1, bit[1]=1, bit[2]=0, bit([3]=1, bit[4]=0, bit[5]=0, bit[6]=0, bit([7]=0

Question 6 : Puissance de deux

☞ En utilisant la **récursivité,** définir la fonction d'entête : **int deuxPuissance(int n)** qui retourne la valeur de 2^n (2 à la puissance n) (on suppose n>=0)

Question 7 : Valeur décimale d'un nombre en binaire

☞ Écrire une fonction de prototype **int decimale(int bit[ ])** ; qui retourne la valeur décimale (base 10) de l'entier représenté en binaire par le tableau **bit** (Le tableau **bit** est supposé être initialisé avec **8** entiers de valeurs **0** ou **1**).

Exemple :

Soit le tableau **bit** initialisé ainsi :
bit[0]=1, bit[1]=0, bit[2]=0, bit[3]=1, bit[4]=0, bit[5]=1, bit[6]=0, bit([7]=0
Alors l'appel de la fonction **decimale(bit)** retourne le nombre **41**.

Question 8 : Opérateur OU EXCLUSIF (XOR)

Soient **x** et **y** deux entiers tels que **(0<= x<=255) et (0<=y<=255)**

☞ Écrire une fonction d'entête **int ouExclusif(int x, int y)** qui retourne le résultat de l'opération **x ou Exclusif (XOR) y** (cette opération est effectuée bit à bit sur les représentations binaires de **x** et **y** selon les règles suivantes :

0 ouExclusif 0 vaut 0
0 ouExclusif 1 vaut 1
1 ouExclusif 0 vaut 1
1 ouExclusif 1 vaut 0

Remarque : Il ne faut pas utiliser l'opérateur ^ (symbole de **XOR** en langage **C**)

Exemple : L'appel de la fonction **ouExclusif (5,6)** retourne **3** (En binaire **5=00000101** , **6=00000110** et **5 OU EXCLIUSIF 6=00000011 =3**)

Question 9 : Algorithme PRGA

☞ Écrire la fonction **void PRGA()** qui permet de générer le texte chiffré en utilisant l'algorithme **PRGA** décrit ci-dessus pour chiffrer la chaine **Claire** en utilisant la clé de chiffrement **Clef**, le texte chiffré sera mis dans la chaine **Chiffree**

B : Chiffrement d'un fichier texte par l'algorithme RC4

Il s'agit de reprendre l'algorithme **RC4** pour chiffrer toutes les lignes d'un fichier texte donné. Pour ce faire on suppose avoir définie la fonction d'entête suivante :

char *RC4Chaine(char *Claire, int L, char *Clef)

Cette fonction a pour paramètres :

- **Claire :** Une chaîne de caractères contenant le texte clair qu'on désire chiffrer
- **L :** Un entier positif contenant la longueur de la chaîne **Claire (0<L<=80)**
- **Clef :** Une chaîne de caractères contenant la clé de chiffrement

La fonction **RC4Chaine** retourne l'adresse d'une chaîne de caractères contenant le texte chiffré par l'algorithme **RC4,** de la chaîne **Claire** en utilisant la clé de chiffrement **Clef.**

Question 10 : Chiffrement d'un fichier

En utilisant la fonction **RC4Chaine** décrite ci dessus,

☞ écrire la fonction de prototype : **void RC4Fichier(char * fich,char *Clef, char * fichChiffre)**. Cette fonction permet de chiffrer en utilisant la chaîne **Clef**($2^{ème}$ paramètre) les lignes du fichier texte de nom physique **fich** (le fichier **fich** en premier paramètre est supposé existant). Les lignes chiffrées du fichier **fich** seront mises dans un nouveau fichier texte de nom physique **fichChiffre** (le fichier **fichChiffre** ($3^{ème}$ paramètre) sera crée dans la fonction)

Remarque : On suppose que les lignes du fichier à chiffrer ont toutes des longueurs inférieures ou égales à 80

Rappels

Quelques prototypes de fonctions de la bibliothèque **stdio.h** utilisant les fichiers de données :

- **FILE *fopen(char *nom, char *m)** : ouvre le fichier **nom** avec le mode d'ouverture **m** (**m="r"** pour la lecture et **m="w"** pour l'écriture) et retourne l'adresse de ce fichier
- **char *fgets(char * ligne, int max, FILE *f)** : lit une ligne du fichier **f**, et la met dans la chaîne **ligne , max** est le nombre maximum de caractères se trouvant dans la ligne à lire. Cette fonction retourne l'adresse de la ligne lue ou **NULL** pour la fin de fichier,
- **int fputs(char * ch, FILE *f) ;** écrit la chaîne de caractères **ch** dans le fichier d'adresse **f**. Elle retourne une valeur positive si l'écriture s'est correctement déroulée ou le caractère **EOF** sinon

✧✧✧

FIN DE L'ÉPREUVE