Programmation C
Examen 1ère session
1er semestre

Durée de l'épreuve : 2 heures

Les notes de cours, de TD et de TP ne sont pas autorisées. Les ordinateurs, calculatrices ainsi que les téléphones sont interdits et éteints durant l'épreuve. Il est possible à tout moment d'admettre la réponse (algorithme, fonction, programme, etc.) à une question non traitée à condition de le préciser explicitement. Il est très fortement préconisé de lire exhaustivement l'énoncé avant de commencer à répondre aux exercices.

Cet examen porte sur votre capacité à la programmation en langage C ainsi qu'à la compréhension que vous avez de ce langage.

Lorsque le comportement d'un code source n'est pas prévisible, c'est à vous de le savoir et de le mentionner clairement sur votre copie. Seules les headers de la bibliothèque standard du langage C sont autorisées (`stdio.h`, `string.h`, `stdlib.h`, ...).

Le barème est indicatif et pourra être changé légèrement lors de la correction.

Exercice 1 Lettres dans un mot (3 points)

Écrire un programme C `lettre` qui compte, par ordre alphabétique, dans tous les arguments qui lui sont donnés dans la ligne de commande les lettres minuscules présentes ainsi que leur nombre d'occurrences. Par exemple, avec l'appel suivant, on devrait obtenir :

```
nborie@perceval:$ ./lettre Bonjour mon cher monsieur Borie
c : 1
e : 3
h : 1
i : 2
j : 1
m : 2
n : 3
o : 5
r : 4
s : 1
u : 2
nborie@perceval:$
```

Les autres caractères (autres que les lettres minuscules) sont donc ignorés.

Exercice 2 **Miroir d'un tableau d'entiers** (3 points)

Écrire une fonction C `void miror_array(int tab[], int n)` qui prend en argument un tableau d'entiers ainsi que sa taille et modifie le tableau de façon que les entrées soient placées comme si on les voyaient dans un miroir. Le bout de code suivant,

```
1 int main(int argc, char* argv[]){
2   int tab[]={12, 45, -3, 18, 5, 7, -50};
3
4   print_array(tab, 7);
5   miror_array(tab, 7);
6   print_array(tab, 7);
7
8   return 0;
9 }
```

devrait donc donner après compilation l'affichage suivant :

```
nborie@perceval:$ ./test
12 45 -3 18 5 7 -50
-50 7 5 18 -3 45 12
```

Exercice 3 **Correction de code** (3 points)

Réécrire une version corrigée du code suivant et donner une commande pour le compiler soigneusement :

```
1  #include <string.h>
2
3  int main(int argc, char* argv){
4    int i;
5
6    if (argc = 1){
7      printf("Pas assez d'arguments\n");
8      return 1;
9    }
10   for (i=strlen(argv[1]) ; i>0 ; i--)
11     printf("%d", argv[1][i]);
12 }
```

Une fois corrigé, ce programme (si on décide de l'appeler `test`) devrait avoir le comportant suivant :

```
nborie@perceval:$ ./test miroir
riorim
nborie@perceval:$
```

Exercice 4 **Fonction récursive** (3 points)

Qu'affiche le programme suivant ? Pourquoi ? Combien d'appels à la fonction `strange` sont-ils exécutés ? Justifiez soigneusement vos réponses.

```
1  #include <stdio.h>
2
3  int strange(int a, int b){
4    if (a >= b){
5      return 1;
6    }
7    return strange(a+1, b) + strange(a, b-1);
8  }
9
10 int main(int argc, char* argv[]){
11   printf("%d\n", strange(0,4));
12   return 0;
13 }
```

Exercice 5 **Remplissage d'une structure** (3 points)

Soit la structure C suivante :

```
1  typedef struct student{
2    char first_name[64];
3    char last_name[64];
4    int age;
5  }Student;
```

Écrire une fonction
`void initialize_student(Student* st, char* first_name, char* last_name, int age)`
qui initialise la variable de type `Student` de manière que les champs soient bien renseignés.

Exercice 6 **Code correcteur simple** (5 points)

On souhaite faire transiter sur un réseau des données sous forme de matrices 7 par 7 d'entiers 0 ou 1. Le problème se situe dans le fait que le canal (vieux fil RJ45 partiellement rongé par Médor et écrasé par un pied de l'armoire) n'est pas complètement sûr. Il arrive qu'un booléen passe du 0 au 1 ou inversement. Ce phénomène est possible mais rare. Ainsi, il n'arrive jamais que deux erreurs se trouvent vraiment dans la même matrice.

$$data = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} & b_{1,6} & b_{1,7} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} & b_{2,6} & b_{2,7} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} & b_{3,6} & b_{3,7} \\ b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} & b_{4,5} & b_{4,6} & b_{4,7} \\ b_{5,1} & b_{5,2} & b_{5,3} & b_{5,4} & b_{5,5} & b_{5,6} & b_{5,7} \\ b_{6,1} & b_{6,2} & b_{6,3} & b_{6,4} & b_{6,5} & b_{6,6} & b_{6,7} \\ b_{7,1} & b_{7,2} & b_{7,3} & b_{7,4} & b_{7,5} & b_{7,6} & b_{7,7} \end{pmatrix}.$$

Pour gérer ces erreurs et même les corriger, on utilise non pas des matrices 7 par 7 mais des matrices avec un booléen de plus par lignes et par colonnes.

$$coded\_data = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} & b_{1,6} & b_{1,7} & p_{1,8} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} & b_{2,6} & b_{2,7} & p_{2,8} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} & b_{3,6} & b_{3,7} & p_{3,8} \\ b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} & b_{4,5} & b_{4,6} & b_{4,7} & p_{4,8} \\ b_{5,1} & b_{5,2} & b_{5,3} & b_{5,4} & b_{5,5} & b_{5,6} & b_{5,7} & p_{5,8} \\ b_{6,1} & b_{6,2} & b_{6,3} & b_{6,4} & b_{6,5} & b_{6,6} & b_{6,7} & p_{6,8} \\ b_{7,1} & b_{7,2} & b_{7,3} & b_{7,4} & b_{7,5} & b_{7,6} & b_{7,7} & p_{7,8} \\ p_{8,1} & p_{8,2} & p_{8,3} & p_{8,4} & p_{8,5} & p_{8,6} & p_{8,7} & p_{8,8} \end{pmatrix}.$$

À la fin de chaque lignes : $p_{i,8}$ vaut 1 s'il y a un nombre impair de boolean valant 1 entre $b_{i,1}, b_{i,2}, \ldots, b_{i,7}$ et $p_{i,8}$ vaut 0 sinon. À la fin de chaque colonne : $p_{8,i}$ vaut 1 s'il y a un nombre impair de booléens valant 1 entre $b_{1,i}, b_{2,i}, \ldots, b_{7,i}$ et $p_{8,i}$ vaut 0 sinon.

$p_{8,8}$ est contraint par sa ligne et sa colonne. Sa valeur dépend de la parité du nombre de booléens valant 1 dans toute la sous-matrice 7 par 7.

On représentera les données codées via un tableau d'entiers à deux dimensions en C: `int[8][8]`.

- Écrire une fonction `int is_valid_matrix(int mat[8][8])` qui retourne 1 si les données sont consistantes et 0 s'il y a une incohérence dans la matrice (par le fil, un booléen a été changé).
- Écrire une fonction `void repair_matrix(int mat[8][8])` qui reconstruit les données de la matrice si cette dernière contenait une seule erreur.
- Que dire si strictement plus d'un booléen pouvaient être détruits/changés dans le fil ?

Exemple de donnée valide

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Une donné avec une erreur dans l'entrée indicée par [4][4].

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \\ 1 & 1 & 1 & 1 & \mathbf{1} & 0 & 1 & 1 & \leftarrow \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & \\ & & & & \uparrow & & & & \end{pmatrix}$$