

# Examen de Programmation I

Sciences et Technologies du Vivant, Semestre 1  
Chimie et Génie Chimique, Semestre 3

Mercredi 8 Février 2006

1. N'oubliez pas de mettre vos nom, prénom et SECTION sur toutes vos copies.
2. Ne rendez pas la donnée.
3. Vous pouvez rédiger vos réponses au crayon.
4. Les transparents du cours sont les seuls documents autorisés. Vous n'avez droit à aucune autre source d'information, y compris les données et corrigés des exercices.
5. Le nombre de lignes de code entre crochets [ ] au début des questions est donné à titre indicatif. Dans le cas des fonctions, les en-têtes et les accolades { et } ne sont pas comptées. Vous n'êtes pas obligés de fournir une réponse comportant *exactement* le même nombre de lignes. Par contre, si votre réponse est *beaucoup plus longue* que celle du corrigé, il est certainement possible de faire plus simple.
6. Les réponses qui font ce qui est demandé mais qui sont inutilement compliquées n'obtiendront pas le maximum des points.
7. Les nombres de points pour chaque question sont donnés dans la marge de droite, et sont à titre indicatif. Le total de points est sur 100.
8. Vous pouvez répondre aux questions dans l'ordre de votre choix, à condition de mettre clairement en évidence la référence de la question.

## Question 1 – Sudoku ..... 30 points

Le principe du jeu Sudoku est de remplir une grille 9×9 similaire à la grille ci-dessous, où certaines cases sont pré-remplies et dont la valeur ne peut plus changer, de façon à ce que : chaque colonne, chaque ligne et chaque carré 3×3 contienne les chiffres de 1 à 9.

Par exemple, pour cette grille, on ne peut pas mettre la valeur 4 dans la première case, puisque 4 est déjà présent sur la première ligne. En revanche, la case sur la 5<sup>eme</sup> ligne, 4<sup>eme</sup> colonne contient forcément 8, puisque toutes les autres valeurs sont impossibles.

	6		1	4		5	
		8	3	5	6		
2							1
8			4	7			6
		6			3		
7			9	1			4
5							2
		7	2	6	9		
	4		5	8		7	

Le but de cet exercice est d'écrire une partie du code d'un programme résolvant les Sudoku.

(a) [5 lignes] Ecrivez la fonction

```
void affiche(int grille[10][10])
```

qui affiche la grille du sudoku contenu dans le tableau `grille`. On n'affichera pas les lignes de séparation, en revanche les cases vides seront affichées comme des points (.).

Pour la grille précédente, on veut donc obtenir :

(6)

```
.6.1.4.5.
..83.56..
2.....1.
8..4.7..6
..6...3..
7..9.1..4
5.....2
..72.69..
.4.5.8.7.
```

On supposera que `grille[i][j]` contient la valeur de la case sur la ligne `i`, colonne `j`. Par commodité, la première ligne et la première colonne auront l'indice 1 : les éléments d'indices 0 ne seront donc pas utilisés. Par exemple, dans le cas de la grille précédente, `grille[1][2]` vaut 6. Une case vide contient la valeur 0 : `grille[1][1]` contient 0.

- (b) Le but de cette question est d'écrire la fonction : (12)

```
void supprime_possibilites(int grille[10][10], bool poss[10][10][10])
```

Comme précédemment, le paramètre `grille` contient les valeurs déjà mises dans la grille (la valeur 0 indique que la case est vide). Cette fonction devra remplir le tableau `poss` de la façon suivante : `poss[i][j][k]` devra contenir `true` si et seulement si on peut mettre la valeur `k` dans la case ligne `i`, colonne `j`, sans entrer en conflit avec une valeur déjà présente dans la grille. Par exemple, après avoir appelé `supprime_possibilites` avec la grille précédente :

```
poss[1][1][6] doit contenir false;
poss[1][1][3] doit contenir true;
poss[1][2][1] doit contenir false.
```

Le début de `supprime_possibilites` consiste à initialiser tous les éléments de `poss` à `true`. Ensuite, pour chaque case déjà remplie, en notant `k` la valeur de cette case, il faut mettre à `false` les éléments `poss[i][j][k]` qui appartiennent à la même ligne, la même colonne ou au même carré  $3 \times 3$  que la case considérée. Il faut également mettre à `false` toutes les possibilités des cases déjà remplies, vu qu'on ne peut plus les changer. Par exemple, le 6 en (1,2) met tous les éléments `poss[1][i][6]` à faux pour `i` de 1 à 9, ainsi que tous les éléments `poss[i][2][6]`, et tous les éléments de troisième indice 6 du carré  $3 \times 3$  en haut à gauche de la grille. Finalement, les éléments `poss[1][2][k]` pour `k` allant de 1 à 9 sont aussi `false`, car la case (1,2) est occupée.

- 1) [4 lignes] Ecrivez la première partie de la fonction qui initialise chacun des éléments du tableau `poss` à `true`.
- 2) [2 lignes] Comment calculer `i0` et `j0`, la ligne et la colonne de la case en haut à gauche du carré  $3 \times 3$  auquel appartient la case `i, j` ?
- 3) [19 lignes] Ecrivez la deuxième partie de la fonction qui met à `false` les éléments du tableau `poss`, où cela est requis, comme expliqué au début de cette question.

- (c) [13 lignes] Ecrivez la fonction (12)

```
bool cherche_coup(bool poss[10][10][10], int * lig, int * col, int * elt)
```

qui parcourt le tableau `poss` pour trouver une case où il n'y a qu'une et une seule possibilité. Les paramètres `lig`, `col` et `elt` permettent de renvoyer la ligne et la colonne de la case, et la valeur à mettre dans cette case.

La fonction devra renvoyer `false` s'il n'existe pas de telle case, `true` sinon.

## Question 2 – Mutations .....25 points

Une molécule d'ADN est une séquence de nucléotides, où les nucléotides possibles sont *adénine*, *guanine*, *thymine* et *cytosine*, respectivement représentés par les 4 lettres A, G, T, et C.

Un triplet de nucléotides est appelé *codon*, et à chaque codon correspond un *acide aminé*. Pour simplifier, on supposera qu'un acide aminé correspond à un seul codon.

Finalement, une *protéine* est une séquence d'acides aminés.

Les séquences ADN seront représentées dans l'exercice par des chaînes de caractères (la fin de la séquence sera indiquée simplement par la fin de la chaîne de caractères, et non pas un codon STOP). On définit également les structures suivantes :

```
struct Acide_amine
{
    char nom[100];
    char codon[3];
};

struct Proteine
{
    char nom[100];
    Acide_amine * acides;
    int nb_acides;
};
```

Les questions ci-dessous sont toutes indépendantes.

- (a) [2 lignes] Supposons qu'on ait déclaré une variable `p` de type `Proteine` : (5)

```
Proteine p;
```

et qu'on ait défini par ailleurs le contenu de ses champs. Ecrivez le code (pas de fonction) qui affiche la liste des noms des acides aminés dont `p` est composée.

- (b) [6 lignes] Ecrivez la fonction : (5)

```
char * proteine_vers_adn(Proteine * p)
```

qui convertit une protéine en chaîne ADN. La fonction prend un pointeur sur `Proteine` en paramètre et retourne une chaîne de caractères contenant la chaîne ADN correspondante. Elle devra bien évidemment se charger d'allouer de la mémoire pour la chaîne de caractères qu'elle retournera.

- (c) [3 lignes] Ecrivez la fonction : (5)

```
Acide_amine * cherche_acide(Acide_amine * acides, int nb_total_acides,
                             char codon1, char codon2, char codon3)
```

où : `acides` est un tableau contenant tous les acides aminés existants, le nombre d'éléments de `acides` étant `nb_total_acides`; et

`codon1`, `codon2` et `codon3` correspondent aux 3 caractères d'un codon.

La fonction devra renvoyer le pointeur sur l'élément de `acides` correspondant au codon fait de `codon1`, `codon2` et `codon3`. On suppose que le codon correspond forcément à un des acides aminés du tableau `acides`.

- (d) [8 lignes] Ecrivez la fonction : (5)

```
Proteine * adn_vers_proteine(char * ADN,
                              Acide_amine * acides, int nb_total_acides)
```

qui crée une protéine à partir de la chaîne de caractères ADN, représentant une chaîne ADN. La fonction devra évidemment allouer la mémoire nécessaire, et renvoyer un pointeur sur la protéine créée. Les paramètres `acides` et `nb_total_acides` ont la même signification que dans la question précédente.

On supposera que la chaîne ADN est correcte (le nombre de caractères est divisible par 3, et chacun des codons est le codon d'un des acides aminés du tableau `acides`).

On pourra utiliser la fonction `cherche_acide` (même si vous n'avez pas su l'écrire).

- (e) [4 lignes] Une mutation consiste en une modification d'une séquence ADN, un ou plusieurs nucléotides étant remplacés par d'autres. (5)

Ecrivez la fonction :

```
char * mutation(char * ADN, int nb_mutations)
```

qui prend une chaîne ADN en paramètre et qui retourne une nouvelle chaîne ADN, similaire à ADN mais après avoir effectué `nb_mutations`, c'est-à-dire après avoir modifié `nb_mutations` nucléotides dont la position sera tirée au hasard, et dont la nouvelle valeur (A, C, G ou T donc) sera également tirée au hasard.

### Question 3 – Pointeurs ..... 30 points

- (a) Indiquez ce qu'affiche le code suivant : (10)

```
int a, b;
int * p, * q;

a = 1;
b = 2;
p = &a;
q = p;
b = *q;
cout << "a1) " << a << " " << b << " " << *p << endl;

a = 1;
p = &a;
b = a;
a = 2;
cout << "a2) " << a << " " << b << " " << *p << endl;

a = 1;
b = 2;
p = &a;
*p = b;
b = a;
a = *p;
cout << "a3) " << a << " " << b << " " << *p << endl;

a = 1;
b = 2;
b = a;
p = 0;
cout << "a4) " << a << " " << b << " " << *p << endl;

a = b;
cout << "a5) " << a << " " << b << " " << *p << endl;
```

- (b) Indiquez ce qu'affiche le code suivant : (10)

```
void f1(int * p, int x)
{
    *p = *p + 1;
    x = x + 1;
}

int f2(int * p)
```

```

{
    *p = *p + 1;

    return *p;
}

int f3(int * p, int x)
{
    *p = *p + 1;
    x = x + 1;

    return x;
}

int f5(int * p, int x)
{
    x = *p;
    *p = *p + 1;

    return x;
}

```

Dans la fonction main :

```

int a, b;
a = 0;
b = 5;
f1(&a, b);
cout << "b1) " << a << " " << b << endl;

a = 0;
b = 5;
b = f2(&a);
cout << "b2) " << a << " " << b << endl;

a = 0;
b = 5;
b = f3(&a, b);
cout << "b3) " << a << " " << b << endl;

a = 0;
b = 5;
a = f3(&a, b);
cout << "b4) " << a << " " << b << endl;

a = 0;
b = 5;
a = f5(&a, b);
cout << "b5) " << a << " " << b << endl;

```

(c) Indiquez ce qu'affiche le code suivant :

```

int T[10];
int * p;

for(int i = 0; i < 10; i++)

```

(10)

```

T[i] = i;

p = T;
cout << "c1) " << *p << endl;

p++;
cout << "c2) " << *p << endl;

p = &(T[2]) + 1;
cout << "c3) " << *p << endl;

p = (T + 5) - 1;
cout << "c4) " << *p << endl;

p = &(T[T[T[3]]]);
cout << "c5) " << *p << endl;

```

**Question 4 – Courbes de Lissajou .....15 points**

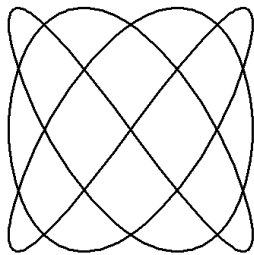


FIG. 1 – La courbe de Lissajou obtenue en utilisant  $a = 3$ ,  $b = 4$  et  $\phi = \pi/2$ .

Ecrivez la fonction :

```

void affiche_lissajou(SimpleWindow * w,
                    int A, float a, float b, float phi,
                    int x0, int y0)

```

qui dessine une *Courbe de Lissajou* dans une fenêtre graphique. Une *Courbe de Lissajou* est une courbe définie par l'équation paramétrique suivante :

$$\begin{cases} x(t) = x_0 + A \sin(at + \phi) \\ y(t) = y_0 + A \sin(bt) \end{cases}$$

où  $t$  varie entre 0 et  $2\pi$ , et  $A, a, b, \phi \in \mathbb{R}$  sont les paramètres.

Par exemple, en utilisant les paramètres  $a = 3$ ,  $b = 4$  et  $\phi = \pi/2$ , on obtient la courbe illustrée par la Figure 1.