

Examen de Programmation I

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

Mercredi 7 Février 2007

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. Cependant, prenez soin d'écrire proprement. Les copies illisibles ne recevront pas de points.
4. 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.
5. Les réponses qui font ce qui est demandé mais qui sont inutilement compliquées n'obtiendront pas le maximum des points.
6. 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.
7. 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 – Élevage de lapins40 points

Dans cet exercice, vous allez écrire du code C permettant de gérer un élevage de lapins. Dans notre élevage moderne, chaque lapin porte un émetteur/récepteur GPS. De cette manière, la position exacte de chaque lapin est connue en tout temps.

- (a) [5 lignes] Dans un premier temps, définissez une structure `Lapin` représentant un lapin. La structure contiendra le nom, l'âge et le poids du lapin, ainsi que sa position dans l'enclos, sous forme de coordonnées x et y . Pour chaque champ de la structure, choisissez le type approprié. La position d'un lapin est définie par des coordonnées entières. (5)

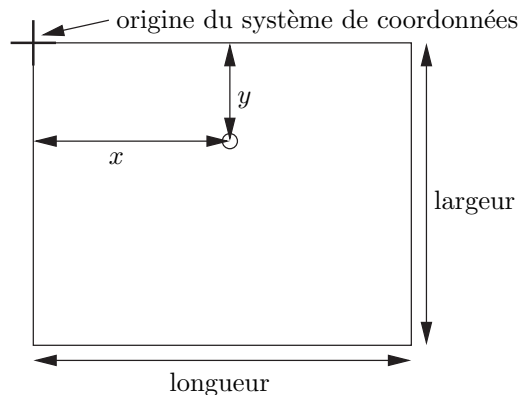


FIG. 1 – Le système de coordonnées d'un enclos.

- (b) [4 lignes] Nous avons également besoin d'une structure pour caractériser un enclos, dans lequel évoluent les lapins. Créez une nouvelle structure appelée `Enclos` qui contiendra un tableau de `Lapins` se trouvant à l'intérieur de l'enclos, une variable indiquant le nombre de lapins, ainsi que deux variables pour stocker la longueur et la largeur de l'enclos. On supposera qu'un enclos a toujours une forme rectangulaire. Ici aussi, prenez soin de choisir le type approprié pour chaque champ. (4)
- (c) [5 lignes] Écrivez une fonction nommée `interieur` permettant de vérifier qu'un lapin se trouve bien à l'intérieur de l'enclos. La fonction recevra un pointeur sur `Lapin` ainsi qu'un pointeur sur `Enclos` et retournera un booléen : `true` si le lapin est à l'intérieur de l'enclos, `false` sinon. La position des lapins – les champs `x` et `y` – est donnée relativement à l'enclos, comme illustré sur la Fig. 1. (6)
- (d) [4-7 lignes] En vous servant de la fonction `interieur`, écrivez une fonction `alarme` qui permet de tester si des lapins se sont échappés de l'enclos. La fonction recevra un pointeur sur `Enclos` et retournera un booléen : `false` si aucun lapin ne s'est échappé, `true` sinon. (6)
- (e) [4 lignes] Nous aimerions maintenant visualiser la position des lapins dans – et éventuellement en dehors de – l'enclos. Pour ce faire, vous allez écrire une fonction utilisant la librairie graphique `SimpleWindow`. Créez une nouvelle fonction avec l'en-tête suivante : `void affiche_enclos(SimpleWindow *win, Enclos *enc, int marge)` (6)

Votre fonction devra afficher l'enclos sous la forme d'un rectangle gris, et les lapins sous forme de points, avec leur nom affiché à côté. Les lapins à l'intérieur de l'enclos seront dessinés en noir et ceux à l'extérieur en rouge. Vous prendrez soin de laisser une marge spécifiée par `marge` autour de l'enclos. Vous pouvez supposer que la fenêtre graphique a les dimensions suffisantes pour que vous puissiez y afficher l'enclos et la marge nécessaire. La Fig. 2 illustre le résultat que devrait afficher votre fonction.

Pour vous simplifier la tâche, on vous donne (vous n'avez pas à l'écrire) la fonction d'en-tête

```
void affiche_lapin(SimpleWindow *win, char *nom, int x, int y, bool noir)
```

qui se charge d'afficher un lapin – un point et le nom – aux coordonnées `x` et `y` de la **fenêtre** (et non pas de l'enclos). Le paramètre `noir` sert à spécifier si le lapin devra être dessiné en noir (`true`) ou en rouge (`false`).

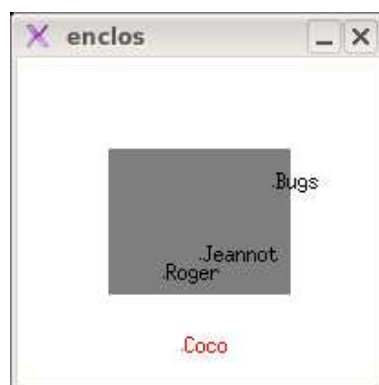


FIG. 2 – Le résultat de la fonction `affiche_enclos`.

Pour mémoire, nous vous rappelons que la librairie `SimpleWindow` possède les fonctions de dessin suivantes :

```
void color(float red, float green, float blue);
void drawPoint(int x, int y);
void drawLine(int x1, int y1, int x2, int y2);
```

```
void drawCircle(int x, int y, int r);
void drawText(char *s, int x, int y);
void fillRectangle(int x, int y, int w, int h);
```

- (f) [9 lignes] Écrivez une fonction d'en-tête (5)
`Lapin *plus_proche_lapin(Enclos *enc, Lapin *lap)`

qui identifie le `Lapin` de l'`Enclos enc` le plus proche du `Lapin lap`. La fonction retourne un pointeur sur le plus proche `Lapin`. On suppose que le `Lapin lap` passé en paramètre appartient bien à l'`Enclos enc` et que `enc` possède au minimum 2 `Lapins`. Pour vous aider, vous pouvez utiliser (vous n'avez pas à l'écrire) la fonction

```
float distance_lapins(Lapin *l1, Lapin *l2)
```

qui retourne la distance entre les deux `Lapins` passés en paramètre.

- (g) [2 lignes] Nous aimerions pouvoir connaître la généalogie de chaque lapin. Dans ce but, modifiez la structure `Lapin` que vous avez écrite au point (a), en lui ajoutant un pointeur sur le père et un pointeur sur la mère du lapin. (3)

- (h) [1 ligne] Écrivez une fonction `affiche_parents` qui affiche la filiation d'un lapin de la manière suivante : (5)
`Jeannot est le fils de Roger et Coco`

La fonction prend comme seul paramètre un pointeur sur le `Lapin` à afficher.

Question 2 – Cryptographie de Boy-Scout 30 points

- (a) [3 lignes] Codage de Jules César : Jules César utilisait un système de codage très simple, qui consiste à remplacer chaque lettre d'un message par la lettre placée plusieurs rangs après ou avant dans l'ordre alphabétique. Par exemple, pour un nombre de rangs de 4, A devient E, B devient F, jusque Z qui devient D. (5)

On demande d'écrire la fonction d'en-tête :

```
void codage_jules_cesar(char * phrase, int n)
```

qui change la chaîne de caractères `phrase` en remplaçant chaque lettre par la lettre placée `n` rangs après dans l'ordre alphabétique. On suppose que `phrase` n'est composée que de lettres majuscules et d'espaces. Les espaces devront rester inchangés. Par exemple, le code :

```
char message[] = "CODAGE DE JULES CESAR";
codage_jules_cesar(message, 4);
cout << message << endl;
```

devra afficher `GSHEKI HI NYPIW GIWEV`. Rappelez-vous que ce type de manipulation est assez facile en C, du moins sur un seul caractère. En effet, après :

```
char char_decale = 'A' + 4;
```

`char_decal` contiendra le caractère E. Attention cependant : le caractère doit rester une lettre : dans l'exemple précédent, Z doit être remplacé par D.

- (b) [12 lignes] Le codage de Jules César est évidemment facile à décrypter. Il suffit bien souvent de trouver la lettre la plus fréquente dans le message codé. Comme dans un texte en français, la lettre la plus fréquente est E la plupart du temps, on en déduit facilement le nombre de rangs utilisé. Dans l'exemple précédent (GSHEKI HI NYPIW GIWEV), I est la lettre la plus fréquente, et il y a bien 4 rangs entre I et E. (10)

On vous demande d'écrire la fonction d'en-tête :

```
char lettre_la_plus_frequente(char * phrase)
```

qui devra retourner la lettre la plus fréquente dans la chaîne de caractères `phrase`.

- (c) [2 lignes] Écrivez la fonction d'en-tête : (4)

```
void decodage_jules_cesar(char * phrase)
```

qui change la chaîne de caractères `phrase` en sa version décryptée, en appliquant l'astuce précédente. Vous pourrez utiliser la fonction `lettre_la_plus_frequente` même si vous n'avez pas su l'écrire, et remarquer que vous pouvez utiliser la fonction `codage_jules_cesar` elle-même pour faire le décodage, une fois que vous avez trouvé le nombre de rangs.

- (d) [3 lignes] Considérons maintenant un codage un peu plus sophistiqué, qui utilise une table de codage. Cette table associe à chaque caractère du message original le caractère à utiliser dans le message codé. Un exemple d'une telle table est : (4)

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
W	P	H	L	B	R	Y	E	T	D	J	K	F	O	V	M	X	U	I	Z	S	N	Q	A	C	G

Si on applique cette table au message `CODAGE TABLE`, on obtient `HVLWYB ZWPKB` comme message crypté. Nous allons utiliser un tableau de caractères pour représenter une telle table, où l'index d'un élément correspond au code ASCII du caractère à coder, et l'élément lui-même est le caractère après codage. Par exemple, si `table` est un tableau contenant la table précédente, `cout << table[int('A')] << endl;` affichera `W`, car `int('A')` retourne le code ASCII de `'A'`.

Nous supposons qu'on dispose déjà d'une telle table. Il n'y a donc pas à la créer.

Écrivez la fonction qui permet de coder un message en utilisant une table déjà créée. La fonction aura pour en-tête :

```
void codage_avec_table(char * phrase, char table[128])
```

et remplacera la chaîne de caractères `phrase` par sa version codée.

- (e) [5 lignes] Écrivez la fonction qui permet de décoder un message. La fonction aura pour en-tête : (7)

```
void decodage_avec_table(char * phrase_codee, char table[128])
```

et remplacera la chaîne de caractères `phrase_codee` par sa version décodée. Le paramètre `table` est le même que précédemment, c'est-à-dire la table de codage.

Question 3 – Pointeurs 30 points

- (a) Indiquez ce qu'affiche le code suivant : (15)

```
void f(int * X, int * Y, int Z)
{
    Z = *X;
    *X = *Y;
    *Y = Z;
}
```

```

void g(int X, int Y, int * Z)
{
    *Z = X;
    X = Y;
    Y = *Z;
}

int main(int argc, char ** argv)
{
    int a = 1, b = 2, c = 3;
    int * p1 = &a, * p2 = &b, * p3 = &c;

    p3 = p2;
    p2 = p1;
    p1 = p3;
    cout << "1a. " << a << " " << b << " " << c << endl;
    cout << "1b. " << *p1 << " " << *p2 << " " << *p3 << endl;

    a = 1; b = 2; c = 3;
    p1 = &a; p2 = &b; p3 = &c;
    *p3 = *p2;
    *p2 = *p1;
    *p1 = *p3;
    cout << "2a. " << a << " " << b << " " << c << endl;
    cout << "2b. " << *p1 << " " << *p2 << " " << *p3 << endl;

    a = 1; b = 2; c = 3;
    f(&a, &b, c);
    cout << "3. " << a << " " << b << " " << c << endl;

    a = 1; b = 2; c = 3;
    g(a, b, &c);
    cout << "4. " << a << " " << b << " " << c << endl;

    a = 1; b = 2; c = 3;
    p1 = &a; p2 = &b; p3 = &c;
    f(p1, p2, *p3);
    g(*p1, *p2, p3);
    cout << "5a. " << a << " " << b << " " << c << endl;
    cout << "5b. " << *p1 << " " << *p2 << " " << *p3 << endl;

    return 0;
}

```

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

(15)

```

struct LDC
{
    int val;
    LDC * g, * d;
};

```

```

void init(LDC * n, int val, LDC * ng, LDC * nd)
{
    n->val = val;
    n->g = ng;
    n->d = nd;
}

int main(int argc, char ** argv)
{
    LDC n1, n2, n3;
    LDC * pn;

    init(&n1, 1, 0, &n2);
    init(&n2, 2, &n1, &n3);
    init(&n3, 3, &n2, 0);

    pn = &n1;
    cout << "A: " << pn->val << endl;
    pn = pn->d;
    cout << "B: " << pn->val << endl;
    pn = (pn->d)->g;
    cout << "C: " << pn->val << endl;

    pn = &n2;
    pn = pn->d;
    pn = pn->g;
    pn = pn->g;
    cout << "D: " << pn->val << endl;

    init(&n1, 1, &n3, &n2);
    init(&n2, 2, &n1, &n3);
    init(&n3, 3, &n2, &n1);

    pn = &n1;
    pn = pn->d;
    pn = pn->d;
    pn = pn->d;
    cout << "E: " << pn->val << endl;

    init(&n1, 1, &n3, &n2);
    init(&n2, 2, &n1, &n3);
    init(&n3, 3, &n2, &n3);

    pn = &n1;
    pn = pn->d;
    pn = pn->d;
    pn = pn->d;
    cout << "F: " << pn->val << endl;

    return 0;
}

```