

Examen de Programmation I

Sciences et Technologies du Vivant, Semestre 1

Mercredi 19 décembre 2007

1. N'oubliez pas de mettre vos NOM et PRÉNOM sur toutes vos copies.
2. Merci de garder 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. Vous n'avez pas à écrire un programme entier avec `#include`, etc. Rédigez uniquement la partie demandée. Dans le cas d'une fonction, écrivez l'en-tête et le corps de la fonction uniquement.
5. Pour l'exercice 3, le nombre de lignes de code entre crochets [] au début des questions est donné à titre indicatif. 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 trop 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. L'examen comporte un maximum de 102 points. La note sera calculée sur 100 points.

Question 1 – Commandes UNIX21 points

Pour cet exercice, on suppose que votre nom d'utilisateur est "username". Vous vous trouvez dans votre "home directory" (/home/username) et celui-ci est initialement vide.

- (a) Veuillez écrire les commandes UNIX nécessaires afin de créer l'arborescence de répertoires illustrée par la Figure 1. (3)

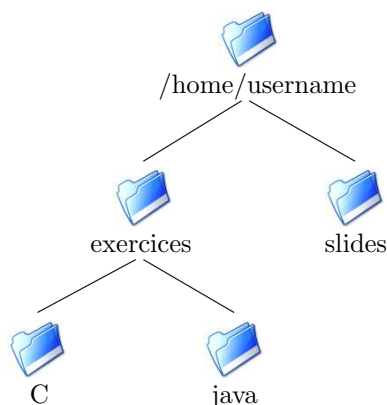


FIG. 1 – Arborescence de répertoires à créer pour la question 1.a

- (b) Dans le répertoire /home/lepetit se trouvent, entre autres, les transparents du cours au format pdf. Ces douze fichiers sont nommés `cours-C-01.pdf` à `cours-C-12.pdf`. Depuis votre "home directory", en **une seule commande** UNIX, copiez les transparents du cours dans le répertoire `slides` que vous avez créé au point précédent. (3)

- (c) On suppose que vous avez passé votre après-midi à faire une série d'exercices et que vous avez placé tous vos programmes dans le répertoire C de la Figure 1. Toujours depuis votre "home directory", écrivez **la** commande qui affiche la liste et les informations des fichiers sources (ceux se terminant par `.cpp`) contenus dans le répertoire C. (3)

Votre commande affichera le résultat suivant :

```
-r--r--r-- 1 username sv-ba1 294 Nov 7 08:38 exercices/C/addition.cpp
-rw-r--r-- 1 username sv-ba1 830 Nov 6 17:03 exercices/C/complation.cpp
-rw-r--r-- 1 username sv-ba1 1234 Nov 6 17:03 exercices/C/complexe.cpp
-rw-r--r-- 1 username sv-ba1 739 Nov 8 10:51 exercices/C/conversion.cpp
-r--r----- 1 username sv-ba1 1270 Nov 6 17:03 exercices/C/date.cpp
-rw-r----- 1 username sv-ba1 563 Nov 6 17:03 exercices/C/palindrome.cpp
-rw-r----- 1 username sv-ba1 2745 Nov 6 17:03 exercices/C/parser.cpp
-rw-r----- 1 username sv-ba1 1201 Nov 6 17:03 exercices/C/pointeurs.cpp
```

- (d) Écrivez **la** commande permettant de vous déplacer de votre "home directory" jusqu'au répertoire C. Pour le reste de la question, vous vous situerez dans le répertoire C. (3)
- (e) En lisant le corrigé de la série, vous vous apercevez que vos programmes `conversion.cpp` et `palindrome.cpp` sont faux. Faites le nécessaire pour effacer ces fichiers. (3)
- (f) Le programme `complation.cpp` est vraisemblablement mal orthographié. Changez son nom en `compilation.cpp`. (3)
- (g) Un camarade d'une autre section (qui n'appartient donc pas à votre *groupe*) aimerait jeter un oeil sur votre programme `pointeurs.cpp`. Devez-vous effectuer un changement pour qu'il puisse le faire? Si oui, lequel? Si non, expliquez. (3)

Indice : référez-vous au listing ci-dessus.

Question 2 – Pointeurs 31 points

- (a) Qu'affiche le programme suivant? (9)

```
void f(int n, int m)
{
    n = m;
    m = n;
}

int main(int argc, char ** argv)
{
    int a = 5, b = 14;
    int *p = &a, *q = &b;

    f(a, b);
    cout << "1) " << a << ", " << b << endl;

    f(*p, *q);
    cout << "2) " << a << ", " << b << endl;

    a = 5; b = 14;
    p = &a; q = &b;
    f(*p, 0);
    cout << "3) " << a << ", " << b << endl;

    return 0;
}
```

(b) Qu'affiche le programme suivant ?

(12)

```
struct Truc
{
    Truc * gauche, * droite;
    int v;
};

Truc cree_truc(Truc * tg, Truc * td, int v)
{
    Truc r;
    r.gauche = tg;
    r.droite = td;
    r.v = v;
    return r;
}

int main(int argc, char ** argv)
{
    Truc a, b, c, d;

    a = cree_truc(0, 0, 1);
    b = cree_truc(0, 0, 2);
    c = cree_truc(&b, 0, 4);
    d = cree_truc(&a, &c, 8);

    cout << "1) " << a.v << " " << b.v << " " << c.v << " " << d.v << endl;

    cout << "2) " << d.gauche->v << " " << d.droite->v << endl;

    cout << "3) " << d.droite->gauche->v << endl;

    a.droite = &b;
    cout << "4) " << d.droite->gauche->v << " "
        << d.gauche->droite->v << endl;

    b.gauche = &d;
    b.droite = &b;
    cout << "5) " << d.droite->gauche->v << " "
        << d.gauche->droite->v << endl;

    cout << "6) " << d.droite->gauche->gauche->v << " "
        << d.gauche->droite->droite->v << endl;

    return 0;
}
```

(c) Qu'affiche le programme suivant ?

(10)

```
int main(int argc, char **argv) {
    const int nb_elements = 6;
    int *p, *q;

    p = new int[nb_elements];
    p[0] = 1;
```

```

for (int i=1; i<nb_elements; i++)
    p[i] = p[i-1] * 2;

cout << "1) ";
for (int i=0; i<nb_elements; i++)
    cout << p[i] << " ";
cout << endl;

q = p + 2;

cout << "2) " << *(p+1) << ", " << *q << endl;

*p += 4;

cout << "3) " << p[1] + q[2] << endl;

q[3] = *p + 1;

q = p + nb_elements - 1;
cout << "4) ";
for (int i=0; i<nb_elements; i++) {
    cout << *q << " ";
    q--;
}
cout << endl;

for (int i=0; i<nb_elements; i++)
    p[i] = nb_elements - i;

q = p + p[nb_elements-1];

cout << "5) " << q[0] << ", " << p[ p[ *q - 1 ] ] << endl;
return 0;
}

```

Question 3 – Flock of Birds50 points

Le but de cet exercice est de comprendre comment le déplacement d'un essaim d'oiseaux peut être simulé. À partir de quelques règles très simples modélisant le déplacement d'un individu, on obtient un comportement de groupe complexe. Ces règles sont :

1. Séparation : Un individu tend à s'éloigner des zones où trop d'individus sont déjà présents.
2. Alignement : Un individu tend à se déplacer dans la même direction que ses voisins.
3. Cohésion : Un individu tend à se rapprocher de ses voisins, s'ils ne sont pas trop nombreux.

Nous allons considérer une version simplifiée, où nous ne tiendrons pas compte de la deuxième règle et où les oiseaux peuvent changer de direction instantanément. Nous allons écrire les fonctions principales pour implanter la simulation. Un individu sera défini par la structure suivante :

```

struct Oiseau
{
    float x, y;
    float dx, dy;
};

```

où x et y désignent les coordonnées dans le plan 2D de l'oiseau, et dx et dy son vecteur vitesse.

Un essaim sera défini par la structure suivante :

```

struct Essaim
{
    Oiseau * oiseaux;
    int nb_oiseaux;
};

```

où `oiseaux` est un pointeur sur un tableau d'Oiseaux, et `nb_oiseaux` le nombre d'éléments de ce tableau.



FIG. 2 – Un essaim d'oiseaux

(a) [4 lignes] Écrivez la fonction (5)

```
Essaim * alloue_essaim(int n)
```

qui permet d'allouer une structure `Essaim` comportant `n` individus.

(b) [5 lignes] Écrivez la fonction (7)

```
void initialise_oiseau(Oiseau * o, int longueur, int hauteur)
```

qui initialise l'individu pointé par `o`. Les coordonnées x et y seront tirées aléatoirement entre 0 et `longueur`, et 0 et `hauteur` respectivement. Le vecteur vitesse défini par les champs `dx` et `dy` devra avoir une norme égale à 2, et une direction aléatoire. Pour cela, on tirera un angle α aléatoirement entre 0 et 2π , et `dx` sera initialisé à $2 \cos \alpha$, `dy` à $2 \sin \alpha$. La valeur de π peut être obtenue avec la constante `M_PI`.

(c) [2 lignes] Écrivez la fonction (7)

```
void initialise_essaim(Essaim * e, int longueur, int hauteur)
qui appelle la fonction initialise_oiseau pour chacun des individus de l'essaim
pointé par e.
```

(d) [18 lignes] Pour pouvoir appliquer les règles de comportement d'un individu, nous allons (12)

avoir besoin du nombre de ses voisins, et de la position moyenne de ses voisins. Deux individus seront considérés comme voisins si leur distance est inférieure à 5.

Écrivez la fonction

```
void regarde_voisinage(Essaim * e, Oiseau * o,
                      int * nb_voisins,
                      float * x_moyen, float * y_moyen)
```

qui détermine le nombre de voisins dans l'essaim `e` de l'individu `o`. La fonction devra également calculer la moyenne des champs `x` des voisins de `o`, et mettre le résultat dans la variable pointée par `x_moyen`. On procédera de même pour les champs `y` et le paramètre `y_moyen`.

Attention à ne pas considérer `o` lui-même comme un de ses voisins, et à ne pas diviser par 0 quand un individu n'a pas de voisins.

(e) [16 lignes] Écrivez la fonction (12)

```
void bouge_oiseau(Essaim * e, Oiseau * o)
```

qui applique les règles 1 et 3 pour modifier les coordonnées `x` et `y` de `o`. Pour cela, on procédera comme suit :

- On appellera tout d'abord la fonction `regarde_voisinage` pour déterminer le nombre de voisins de `o` et leur position moyenne.
- Si le nombre de voisins vaut 0, on ne change pas `dx` et `dy`.
- Si le nombre de voisins est supérieur à 5, c'est la première règle qui s'applique, et l'individu doit s'éloigner de la position moyenne de ses voisins. Pour cela, les champs `dx` et `dy` seront modifiés de la façon suivante :

$$\begin{pmatrix} dx \\ dy \end{pmatrix} = \frac{2}{N} \begin{pmatrix} x - x_{\text{moyen}} \\ y - y_{\text{moyen}} \end{pmatrix}$$

où N est la norme du vecteur :

$$\begin{pmatrix} x - x_{\text{moyen}} \\ y - y_{\text{moyen}} \end{pmatrix}$$

- Si le nombre de voisins est compris entre 1 et 5, c'est la troisième règle qui s'applique, et l'individu doit s'approcher de la position moyenne de ses voisins. Pour cela, les champs `dx` et `dy` seront modifiés de la façon suivante :

$$\begin{pmatrix} dx \\ dy \end{pmatrix} = -\frac{2}{N} \begin{pmatrix} x - x_{\text{moyen}} \\ y - y_{\text{moyen}} \end{pmatrix}$$

- Dans tous les cas, on ajoutera la valeur courante des champs `dx` et `dy` aux champs `x` et `y`.

(f) [2 lignes] Finalement, écrivez la fonction (7)

```
void affiche_essaim(SimpleWindow * window, Essaim * e)
```

qui dessine chacun des individus de `e` dans la fenêtre pointée par `window`. Un individu sera simplement représenté par un point grâce à la fonction `drawPoint`.