# Examen C++/Biomod, UE 4TBI806U

*Durée de l'épreuve : 3h, Date : jeudi 16 avril 2020, 14h*

A la fin de l'épreuve vous devez chacun me renvoyer normalement 2 ou 3 fichiers C++ (.cpp) fonctionnels et commentés en fichiers attachés par email à **andre.garenne@u-bordeaux.fr** et attendre d'avoir reçu la réponse de ma part comme quoi je les ai bien reçus.

---

Pour l'épreuve vous avez le choix **uniquement** deux possibilités :

- **soit** 3 exercices choisis parmi les 4 premiers
- **soit** l'exercice 5 et un autre au choix

L'exercice 5 est en effet un peu plus long que les autres et l'exercice 1 est le plus court.

**Prenez bien le temps de lire les énoncés avant de faire vos choix !!!**

---

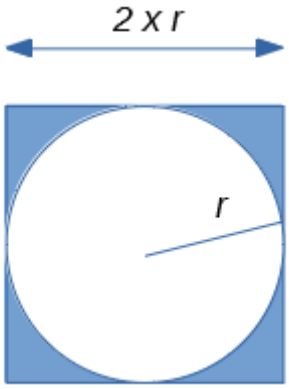
J'aurai beaucoup de fichiers à traiter donc pensez bien à mettre au début de chacun en commentaire vos **nom**, **prénom** et **numéro d'exercice**. Exemple :

```
// Nom, Prénom
// ex_01.cpp
... votre code ...
```

Comme cet examen devrait se dérouler en conférence audio vous pouvez éventuellement me poser des questions d'ordre général durant la séance. Si vous préférez des échanges privés vous pouvez passer par les emails. Je serai aussi réactif que possible.

# ex_01.cpp

Ecrire un programme capable de donner une valeur approchée du nombre $\pi$. Pour ce faire vous utiliserez la technique suivante. Vous considérez une surface carrée dans laquelle s'inscrit un cercle.




Le carré aura une largeur de 2 et le rayon du cercle est de 1. Vous générez ensuite $N$ tirages aléatoires uniformément répartis sur la surface du carré et définis par une position $x_i$ et $y_i$ à chaque essai. Si le point est situé **dans le cercle** il est comptabilisé dans la variable `total_cercle`. Au bout de $N$ essais le rapport de $\frac{total\_cercle}{N}$ doit vous permettre de calculer facilement une estimation de la valeur de $\pi$. Vous ne devrez bien entendu utiliser aucune fonction trigonométrique.

Vous devriez obtenir par exemple un résultat de ce type pour N=100000 :

```
Estimation de pi : 3.14568
```

# ex_02.cpp

Ecrire un programme qui permet de jouer au pendu. Le programme devra lire le fichier `mots.txt` situé dans le même dossier, choisir aléatoirement un des mots situés dedans et dérouler une partie unique. Quand vous pensez avoir trouvé la solution vous entrez un `?` et le programme vous demande alors d'entrer votre solution. Exemple :

```
Essai : 1
---------
Entrer une lettre : a
Essai : 2
---------
Entrer une lettre : e
Essai : 3
-e--e---e
Entrer une lettre : i
Essai : 4
-e--e---e
Entrer une lettre : o
Essai : 5
-e--e---e
Entrer une lettre : u
Essai : 6
-e--e---e
Entrer une lettre : n
Essai : 7
-e--e---e
Entrer une lettre : m
Essai : 8
-e--e---e
Entrer une lettre : s
Essai : 9
-e--es--e
Entrer une lettre : r
Essai : 10
-erres-re
Entrer une lettre : ?
Entrer votre solution : terrestre
Bravo, trouve en 10 essais
```

Remarque 1 : pour éviter que le programme choisisse toujours le même mot à chaque lancement du programme il est possible d'initialiser "aléatoirement" le générateur de nombres aléatoires en le calant sur l'horloge interne de la façon suivante :

```cpp
#include <iostream>
#include <string>
#include <fstream>
#include <random>
#include <chrono>
using namespace std;

default_random_engine dre;

...
```

```
int main(){
    dre.seed(std::chrono::system_clock::now().time_since_epoch().count());
    ...
}
```

Remarque 2 : pour savoir comment lire le contenu d'un fichier texte vous pouvez réutiliser, en l'adaptant ici pour l'occasion, le programme donné à la page 52 du cours :

```
// code_28.cpp
#include <iostream>
#include <string>
#include <fstream>
using namespace std; // utilisation de l'espace de nom de std
int main(){
    // ouverture en lecture
    ifstream fichier("test_27.txt"); // , ios::in
    string data; // pour stocker le contenu du fichier
    string line;
    if(fichier){
        while(getline(fichier, line)){
            data+=line+"\n"; // \n est ajouté car le saut de ligne est omis
        }
        fichier.close();
    } else cerr << "Impossible d'ouvrir le fichier !" << endl;
    cout << data;
    return 0;
}
```

Remarque 3 : quand vous appliquez la fonction `at(i)` à une chaine de caractère, ce qui vous est renvoyé est un `char` et pas une chaine de longueur 1 ce qui implique qu'il ne pourra être comparé qu'à un autre `char`.

# ex_03.cpp

En utilisant **strictement** le prototype ci-dessous :

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
using namespace std;

void echange_01(int& a, int& b){
    ...
}

void echange_02(int* a, int* b){
    ...
}

void sort(vector<int>& to_sort){
    ...
}

int main(){
    int a=10;
    int b=15;
    cout << "a : " << a << " et b : " << b << endl;
    echange_01(a,b);
    cout << "echange_01(a,b)" << endl;
    cout << "a : " << a << " et b : " << b << endl;
    echange_02(&a,&b);
    cout << "echange_02(&a,&b)" << endl;
    cout << "a : " << a << " et b : " << b << endl;
    vector<int> c;
    ... // affichage du contenu de la variable c en ligne
    sort(c);
    ... // affichage du contenu de la variable c en ligne une fois trié
    return 0;
}
```

écrire un programme qui réalise le travail suivant une fois exécuté :

```
a : 10 et b : 15
echange_01(a,b)
a : 15 et b : 10
echange_02(&a,&b)
a : 10 et b : 15
Entrer un entier >0 (0 pour finir) : 12
Entrer un entier >0 (0 pour finir) : 6
Entrer un entier >0 (0 pour finir) : 1
Entrer un entier >0 (0 pour finir) : 5
Entrer un entier >0 (0 pour finir) : 19
Entrer un entier >0 (0 pour finir) : 13
Entrer un entier >0 (0 pour finir) : 3
Entrer un entier >0 (0 pour finir) : 0
c : 12 6 1 5 19 13 3
```

```
c (trie) : 1 3 5 6 12 13 19
```

**Attention** : ce programme doit donc être capable de réaliser un tri par ordre croissant **aux conditions suivantes** :

- le contenu de la variable `vector` **doit être directement modifié** (pas de return)
- la fonction `sort()` **doit utiliser** la fonction `echange_01()` définie avant
- bien entendu vous n'utiliserez pas des fonctions de tri existantes mais vous pouvez vous inspirer de ce que vous trouverez sur internet si vous n'avez aucune idée d'algorithme. **Dans ce cas mentionnez le lien URL de votre source**.

# ex_04.cpp

Un modèle proie-prédateur simple suit les équations de Lotka-Volterra. Si $x$ représente les proies (en milliers) et $y$ les prédateurs (en milliers) la dynamique du système est donnée par :

$$\frac{dx(t)}{dt} = x(t) \times (\alpha - \beta \times y(t))$$

$$\frac{dy(t)}{dt} = y(t) \times (\delta - \gamma \times x(t))$$

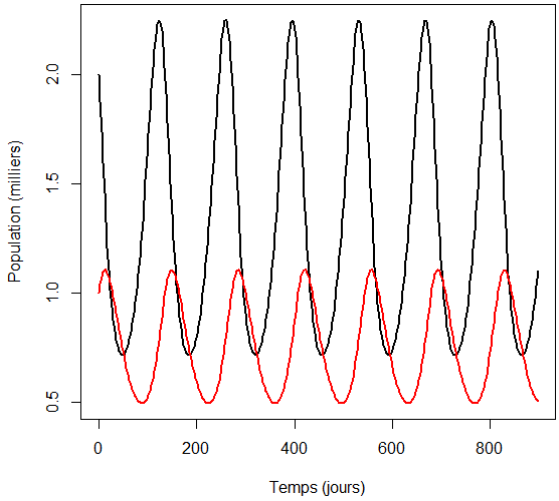
Ecrire un programme qui permette de simuler ce système sur une durée de 30 mois (`duration=900`) avec un pas de temps `dt` de 3 jours (dans ce cas chaque mois contient 30 jours). $x_{(t=0)}$ et $y_{(t=0)}$ sont les nombres de milliers d'individus concernés et sont respectivement de 2 et 1. Après la simulation votre programme doit sauvegarder les données dans un fichier en 3 colonnes : jour, proies, predateurs.

Les paramètres connus sont $\alpha = 0.0667,\ \beta = 0.0875,\ \delta = -0.0333,\ \gamma = -0.025$

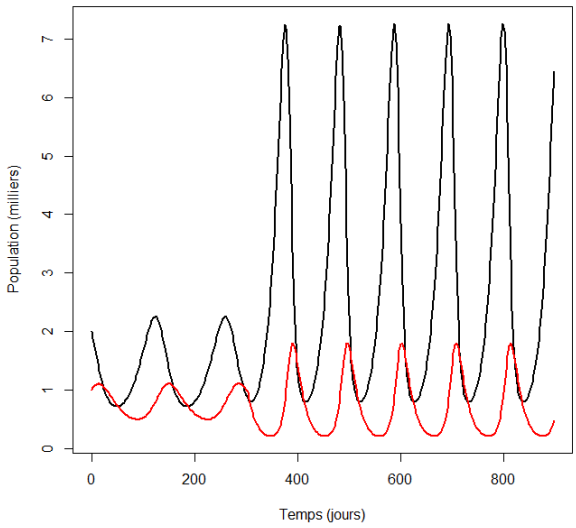
A l'aide du script R suivant exécuté dans le même répertoire :

```
data_1=read.table("result.txt",header=T)
matplot(data_1[,1],data_1[,2:3],type="l",lty=1,lw=2,xlab="Temps
(jours)",ylab="Population (milliers)")
```

Vous devriez obtenir un graphe similaire à celui-ci ou les prédateurs figurent en rouge et les proies en noir :




Maintenant vous allez ajouter un évènement. A partir du 10ème mois une maladie décime 10% des prédateurs tous les 3 jours. Vous devriez avec le même script R observer un graphe de ce type :




- Expliquez brièvement en commentaire ce que vous observez comme interrelations entre proies et prédateurs sur la courbe et essayez d'expliquer ce que vous observez sur la seconde figure.
- Quelles sont selon vous les limites de ce modèle ?

Les réponses doivent figurer en commentaire dans votre fichier cpp.

# ex_05.cpp

Vous devez simuler une épidémie au sein d'une population selon les modalités suivantes. La population étudiée contient 1000 personnes. Chaque personne peut être dans l'un des 3 états :

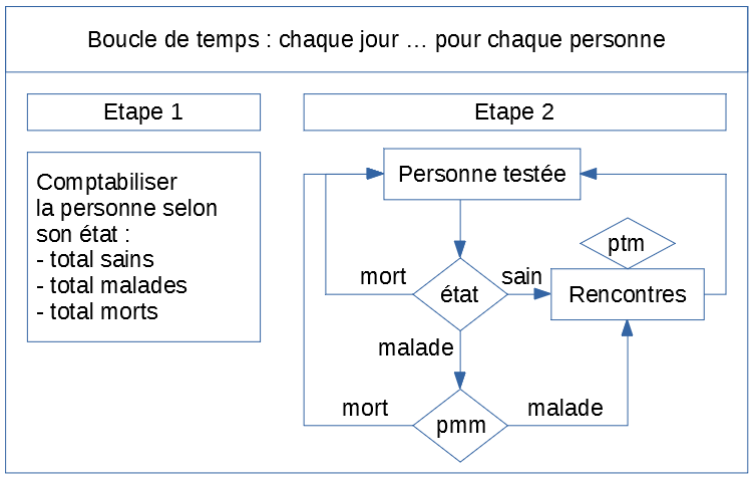
- 0 - sain
- 1 - malade et contagieux
- 2 - mort

### Scénario 1

Vous allez réaliser une simulation sur une durée de 1 an avec un pas de temps de 1 jour. Chaque jour chaque personne est testée et selon son état le nombre de malades, morts et personnes saines est augmenté. Ensuite, chaque personne est testée pour plusieurs évènements consécutifs :

- chaque jour, une personne malade a une chance `pmm=0.001` de décéder.
- chaque jour, une personne va côtoyer un nombre de personnes aléatoire `max_rencontres` entre 1 et 50 tirées au hasard dans la population. On considère des tirages avec remise. Si la personne choisie au hasard est morte, la rencontre n'a simplement pas lieu. A chacun de ces contacts correspond une probabilité de transmettre la maladie `ptm=0.0003` si et uniquement si (i) soit la personne concernée est malade et la rencontre est "saine", (ii) soit l'un des contacts est malade et la personne concernée est "saine". Une personne peut donc potentiellement être contaminée et contaminer en retour chacune des personnes rencontrée chaque jour.

Le schéma ci-dessous résume les évènement possibles et les étapes pour chaque pas de temps et chaque personne :




Vous pourrez utiliser le script R suivant pour visualiser le résultat :

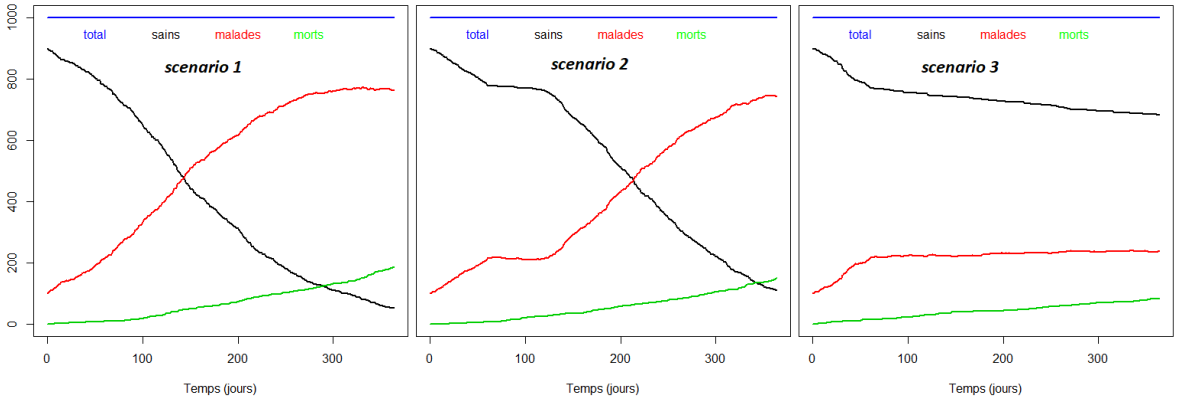
```
data_1=read.table("record.txt",header=T)
matplot(data_1[,1],data_1[,2:5],type="l",lty=1,lw=2,xlab="Temps
(jours)",ylab="Population")
text(50,950,"total",col="blue")
text(125,950,"sains",col="black")
text(200,950,"malades",col="red")
text(275,950,"morts",col="green")
```

### Scénario 2

Vous étudierez ensuite le cas où un confinement de deux mois a lieu de j 60 à j 120. Dans ce cas le paramètre modifié est `max_rencontres` qui est divisé par 10.

### Scénario 3

Enfin vous testerez l'hypothèse où, **en plus du confinement entre j 60 et j 120**, chaque membre de la population porte un masque **après** le déconfinement. Le paramètre supplémentaire modifié est alors `ptm` dont la valeur est divisée par 10 là aussi. Vous devriez observer ce type de graphique en fonction des scénarios :




Pour les exercices **ex_04.cpp** et **ex_05.cpp** vous pouvez me renvoyer, en plus des programmes et de vos éventuels commentaires, les graphes que vous obtenez sous la forme de fichiers png ou jpg de préférence.

Remarque : pour gagner en lisibilité vous pouvez réaliser successivement deux étapes à chaque pas de la boucle de temps :

1. comptage de l'effectif chaque sous population (sain, mort, malade) pour ajouter les totaux aux données à sauvegarder
2. réalisation des différents évènement pouvant changer ou pas le statut de chaque individu

Mais ce n'est qu'une suggestion.