

## TD 3

Programmation sécurisée en C++ moderne	Enseignant	Chiheb Ameer ABID
--	------------	-------------------

### Exercice 1.

Écrire la classe Fraction pour permettre le code suivant :

```
#include <cassert>
auto main() -> int {
    Fraction f0 {5}; // f0=5/1
    assert(f0 == 5 && "5/1 équivaut à 5.");
    Fraction f1 { 5, 2 };
    assert(f1 == 2.5 && "5/2 équivaut à 2.5.");
    f1.simplifier();
    assert(f1.numerateur==5 && f1.denominateur==2 && "5/2 après
simplification donne 5/2.");
    f1 = pow(f1,2);
    assert(f1.numerateur == 25 && f1.denominateur == 4 && "5/2 au carré
équivaut à 25/4.");
    Fraction f2 { 258, 43 };
    assert(f2 == 6 && "258/43 équivaut à 6.");
    f2.simplifier();
    assert(f2.numerateur == 6 && f2.denominateur == 1 && "258/43 après
simplification donne 6/1.");
    f2 = pow(f2, 2);
    assert(f2.numerateur == 36 && f2.denominateur == 1 && "6/1 au carré
équivaut à 36/1.");
    return 0 ;
}
```

Votre classe devra notamment gérer :

- Un constructeur acceptant un entier (créant une fraction de type  $n/1$ ),
- Un constructeur acceptant un numérateur et un dénominateur,
- Un opérateur de conversion (implicite et explicite) au type **double**
- Une méthode **simplifier()**,
- La fonction **pow(Fraction, int)** renvoyant une nouvelle fraction

Soit la fonction **int foo(Fraction)**;

Interdire l'appel à **foo(5)**;

### Exercice 2.

- Écrire une fonction **fois\_deux** qui renvoie le double de son argument entier.
- Écrire une fonction **fact** qui renvoie la factorielle de son argument entier.
- Écrire une fonction **void applique\_tableau(std::function<int(int)> f,int \*tab,int n)**; qui modifie chaque élément du tableau **tab** de taille **n** en son image par la fonction pointée par **f**.
- Soit un tableau d'entiers **tab**. Écrire une suite d'instructions qui modifie chaque élément du tableau en le double de sa factorielle. On suppose que les valeurs du tableau sont telles que chaque calcul peut se faire sans dépassement de capacité. On demande d'écrire deux fonctions lambdas : une pour calculer le double et l'autre pour calculer la factorielle.

### Exercice 3.

Réécrire les deux fonctions de tri du code ci-après en une seule fonction générique de tri, en utilisant :

- une fonction lambda passée en paramètre pour définir la comparaison,
- un objet **std::function<>** pour stocker cette lambda.

Proposer également une fonction lambda pour afficher chaque fois les éléments d'un tableau.

```
void tri_croissant(int *t ,int n) {
    int i,i_min,j,tmp ;
    for (i=0;i<n-1;i++) {
        i_min=i ;
        for (j=i;j<n;j++) {
            if (t[j]<t[i_min]) {
                i_min=j;
            }
        }
        tmp=t[i_min];
        t[i_min]=t[i];
        t[i]=tmp;
    }
}

void tri_decroissant(int*t ,int n) {
    int i,i_max,j,tmp;
    for(i=0;i<n-1;i++) {
        i_max=i;
        for(j=i;j<n;j++) {
            if (t[j]>t[i_max]) {
                i_max=j;
            }
        }
        tmp=t[i_max];
        t[i_max]=t[i];
        t[i]=tmp;
    }
}

int main() {
    int t1[] {5,3,8,6,2,7,4,1};
    int n1=sizeof(t1)/sizeof(t1[0]);
    tri_croissant(t1,n1);
    for (int elt : t1) {
        std::cout<<elt<<" ";
    }
    std::cout<<std::endl;

    int t2[] {5,3,8,6,2,7,4,1};
    int n2=sizeof(t2)/sizeof(t2[0]);
    tri_decroissant(t2,n2);
    for (int elt : t2) {
        std::cout<<elt<<" ";
    }
    std::cout<<std::endl;
    return 0;
}
```