

## TP 2: Types composés

Lionel Lacassagne

### Introduction

Le but de ce TP est, à travers des exercices de complexités croissantes de mettre en oeuvre les concepts de base du C. Pour ce premier TP, un squelette de programme est disponible à l'adresse [www.ief.u-psud.fr/~lacas/Teaching](http://www.ief.u-psud.fr/~lacas/Teaching). A noter que le fichier `param_perl.h` est le fichier pour les tests unitaires automatiques. Il est configuré de manière à être transparent durant la phase de mise au point (tous les `#define` sont activés).

### 1 Tableaux

Dans toute la suite, les tableaux considérés sont des tableaux statiques de taille `N`, une constante, et de type `int`. Afin d'être compatible avec n'importe quelle taille, les fonctions prendront comme taille de tableau le paramètre `n`. Le nom des fonctions de test est le nom de la fonction implantant un algorithme préfixée par `test_`.

1. Coder la fonction qui initialise un tableau à zéro. Nom de la fonction: `tableau_zero()`.
2. Coder la fonction qui initialise un tableau par une suite croissante ou décroissante. Nom de la fonction: `tableau_init_param()`. Les paramètres sont `x0` la première valeur du tableau et `xstep` l'incrément à ajouter à chaque case.
3. Coder la fonction qui initialise un tableau par un générateur de nombres pseudo-aléatoire à congruence linéaire. Nom de la fonction: `tableau_init_random()`. Les paramètres sont `a`, `x0`, `b` et `m`. Avec `x0` la première valeur de la suite et les autres paramètres permettant de calculer les autres termes de la suite telle que :  $x_{n+1} = (a \times x_n + b) \bmod m$ .
4. Coder la fonction d'affichage de tableau `tableau_display` qui prend comme arguments le tableau `T`, sa taille `n`, une chaîne de formatage des affichages `format` du style `"%3d"` et `nom` le nom du tableau. Deux cas particuliers sont traités. Si `format` est égal à `NULL` la chaîne est `"· · · %d · · ·"` (avec un espace avant et après). Si `nom` est égal à `NULL` il n'est pas affiché. Fonction déjà codée.
5. Coder la fonction de tri par sélection, triant de tableau par ordre croissant, et le parcourant de gauche à droite. Nom de la fonction: `tableau_tri_selection()`.
6. Coder la fonction de tri par sélection, triant de tableau par ordre croissant, mais en le parcourant de droite à gauche (on commence par trier le plus grand élément et non le plus petit). Nom de la fonction: `tableau_tri_selection_reverse()`.
7. Coder la fonction de tri par insertion, triant de tableau par ordre croissant, et le parcourant de gauche à droite. Nom de la fonction: `tableau_tri_insertion()`.

8. Coder la fonction de tri par insertion, triant de tableau par ordre croissant, mais en le parcourant de droite à gauche (on commence par trier le plus grand élément et non le plus petit). Nom de la fonction: `tableau_tri_insertion_reverse()`.
9. Coder la fonction trouvant la plus grande valeur du tableau. Nom de la fonction: `tableau_max()`.
10. Coder la fonction trouvant la seconde plus grande valeur du tableau. Nom de la fonction: `tableau_max2()`.

## 2 Fractions

Dans cet exercice on souhaite implanter les fonctions arithmétiques de base permettant de manipuler des fractions dont le numérateur et le dénominateur sont codés sur 16 bits (voir fichier `fraction.h`). Les développements demandés ici sont à but pédagogique. Ils seront améliorés dans un autre TP avec la notion de pointeur.

La fonction `gcd` (*Greatest Common Divisor*) permet de réduire toute fraction réductible en une fraction irréductible en divisant le numérateur et le dénominateur par leur PGCD. Afin d'éviter les erreurs de débordement de capacités (*overflows*), les calculs intermédiaires des fonctions arithmétiques et du PGCD seront fait dans des entiers 32 bits signés. Ainsi la fraction 120/150 se réduit à 4/5, car  $120 = 2^3 \cdot 3 \cdot 5$ ,  $150 = 2 \cdot 3 \cdot 5^2$  et donc  $gcd(120, 150) = 30 = 2 \cdot 3 \cdot 5$ .

1. Modifier la fonction `gcd` pour que le résultat soit toujours positif. La fonction mathématique calculant la valeur absolue d'un entier est `abs()`.
2. Fonction `fraction_display`: déjà codée. Traitement des deux mêmes cas particuliers que dans l'exercice précédent.
3. Coder la fonction `fraction_zero()` qui initialise une fraction à zéro. ‘
4. Coder la fonction `fraction_fraction(a, b)` qui construit une fraction à partir de deux entiers 16 bits signés. Cette fonction doit réduire la fraction si nécessaire et doit toujours renvoyer une fraction dont le numérateur est positif.
5. Coder la fonction `fraction_copy()` qui construit une fraction à partir d'un fraction. Cette fonction doit réduire la fraction si nécessaire et doit toujours renvoyer une fraction dont le numérateur est positif.
6. Coder la fonction `fraction_add()` qui additionne deux fractions et renvoie la fraction réduite.
7. Coder la fonction `fraction_sub()` qui soustrait deux fractions et renvoie la fraction réduite.
8. Coder la fonction `fraction_mul()` qui multiplie deux fractions et renvoie la fraction réduite.
9. Coder la fonction `fraction_div()` qui divise deux fractions et renvoie la fraction réduite.