Tout ce que vous avez toujours voulu savoir sur les pointeurs...

(sans jamais oser le demander)

1 Consignes de rendu

A la fin de ce TP, vous serez dans l'obligation de rendre une archive respectant l'architecture suivante :

```
-- rendu-tpcs11-login_x.zip
   |-- login_x/
       |-- AUTHORS
       |-- README
       |-- BONI.txt
       |-- Unsafe
           |-- Unsafe.cs
           |-- safe.dll
       |-- Unsafe.sln
```

Bien entendu, vous devez remplacer `login_x` par votre propre login.

N'oubliez pas de vérifier les points suivants avant de rendre :

Fichier AUTHORS

Le fichier nommé `AUTHORS` doit se trouver à la racine de votre rendu. Il devra contenir tous les auteurs du projet (en l'occurrence, VOUS seulement). Le format, pour chaque auteur, est le suivant :

— Le caractère '*' suivi d'un espace.
— Le login de l'étudiant.
— Un retour à la ligne.

Fichier README

Le fichier nommé `README` doit se trouver à la racine de votre rendu. Il devra contenir vos remarques à propos de ce TP et de votre travail.

Conseils et remarques

Lisez le sujet dans son intégralité avant de commencer. Une fois que vous l'avez lu, relisez-le. Réalisez les exercices dans l'ordre car ils sont en difficulté croissante.
N'oubliez pas de vérifier les points suivants avant de rendre :

— Le fichier `AUTHORS` doit être au format habituel (rappelé ci-dessus).
— Aucun dossier `bin` ou `obj` dans le rendu.
— Les fonctions demandées doivent respecter le prototype donné
— **Le code doit COMPILER**.
— Si vous faîtes des boni, n'oubliez pas de l'écrire dans un fichier nommé `BONI.txt`

Lorsque cela n'est pas précisé, vous êtes totalement libre quant à l'implémentation des fonctions.

2 Cours

Les pointeurs est un concept important en programmation et même si leur utilisation est rarement nécessaire en C#, ils peuvent être utiles dans certaines situations :

— travail sur des structures existantes sur disque ;
— scénarios COM ou d'appel de plate-forme avancés requérant à des structures contenant des pointeurs ;
— code à performances critiques.

L'utilisation de contexte unsafe dans d'autres situations n'est pas recommandée. En particulier, il ne faut pas utiliser un contexte unsafe pour tenter d'écrire du code C en C#.

Exemple	Description
int* p	p est un pointeur vers un entier.
int** p	p est un pointeur de pointeur vers un entier.
int*[] p	p est un tableau unidimensionnel vers des entiers.
char* p	p est un pointeur vers un caractère.
void* p	p est un pointeur vers un type inconnu.

Attention :
L'expression suivante déclare trois pointeurs vers des entiers.

```
int* a, b, c;
```

3 Petits tests entre caractères

Tout d'abord, pour pouvoir utiliser les pointeurs en C#, il vous faut autoriser le code `unsafe` : pour cela, allez dans les Propriétés de votre projet, puis `Build` et cochez `Allow unsafe code`. Vous pouvez désormais utiliser le mot clé `unsafe`.

Printons !

Commençons par écrire une fonction d'impression `print`, avec deux surcharges : une prend un caractère, l'autre un pointeur vers un caractère.

Prototypes :

```
static unsafe void print(char a);
static unsafe void print(char* a);
```

Les deux fonctions imprimeront dans la console la phrase suivante :

```
The char [1] is located at [2].
```

avec [1] étant la valeur du caractère en question et [2] son pointeur.

Késako ?

Ajoutons maintenant cette fonction, et appelons-la dans le `main`.

```
static unsafe void test_char()
{
        char a = 'a';
        print(&a);
        print(a);
}
```

Exécutez votre code... Les résultats sont différents, pourquoi ?
Écrivez un commentaire au dessus de cette fonction pour expliquer le phénomène.

Swap

Ecrivez maintenant une fonction `swap` qui va échanger les deux valeurs pointées par `p` et `q`. Son prototype est le suivant :

```
public static unsafe void swap(int* p, int* q);
```

4 Opérations sur les tableaux

Tout d'abord, téléchargez le fichier `safe.dll` sur `perso.epita.fr`. Placez le fichier dans le dossier de votre projet (se référer aux consignes de rendu pour plus de précision) puis ajoutez le dll à votre projet :

— Dans le Solution Explorer (à droite), cliquez droit sur `References` ;
— Choisissez "Add Reference" ;
— Puis "Browse" et validez votre dll.

Maintenant que la bibliothèque est incluse dans votre projet, ajoutez le `using` correspondant tout en haut de votre fichier.

Cette bibliothèque vous donne accès à deux classes : `Unsafe` et `Misc`. `Unsafe` contient la version corrigée des fonctions que vous devrez implémenter : vous pourrez ainsi tester votre TP ! De plus, vous pouvez utiliser les fonctions de `Misc` pour générer des tableaux aléatoires.

Rappel :
Une string est simplement un tableau de caractères. Vous pouvez donc faire sur une string les mêmes opérations que sur les tableaux.

Impression d'un tableau

Écrivez la fonction `print_array()` qui prend en paramètre un pointeur d'entier et un entier représentant la taille. Cette fonction va parcourir `size` entiers du tableau et les imprimer, puis revenir à la ligne.

Prototype :

```
static unsafe void print_array(int* src, int size);
```

Écrivez maintenant deux surcharges de cette même fonction mais qui prendra cette fois seulement un pointeur de caractère. En suivant le même principe que la fonction précédente, vous devez maintenant parcourir le tableau de caractères jusqu'au dernier élément, puis revenir à la ligne. De plus, elle devra retourner un entier correspondant au nombre de caractères affichés.
Conseil : le caractère `\0` indique la fin de la chaîne.

Prototype :

```
static unsafe int print_array(char* src);
```

Modification d'un tableau

Écrivez la fonction `fill_array()` qui prend en paramètre un pointeur de caractère et un caractère, et remplit le tableau avec ce dernier. De plus, elle devra retourner un entier correspondant au nombre de caractères modifiés.

Prototype :

```
static unsafe int fill_array(char* src, char val);
```

Strlen

La fonction `strlen()` retourne un entier correspondant à la taille du tableau de caractères.

Prototype :

```
static unsafe int strlen(char* src);
```

Clone

La fonction `clone()` duplique le tableau de caractères source `src` et renvoie le pointeur sur son premier.

Prototype :

```
static unsafe char* clone(char* src);
```

ROTn

Ici, vous allez devoir écrire deux fonctions `rot()` (vous devriez les connaître parfaitement) : elles doivent effectuer une rotation de `n` caractères sur la source passée en argument et renvoyer le résultat.
La source est un `char` dans le premier cas et un `char*` pour la deuxième fonction !
Conseil : écrire la première fonction en premier peut être utile...

Prototype :

```
static char rot(char src, uint n);
static unsafe void rot(char* src, uint n);
```

Remove

La fonction `remove()` supprime le caractère à la position `index` et décale tous les entiers suivants. Le dernier caractère doit être `\0`.

Prototype :

```
static unsafe void remove(char* src, int index);
```

Conversion

La fonction `to_string()` convertit un tableau de caractères en string.

Prototype :

```
unsafe string to_string(char* src);
```

Memcpy

La fonction `memcpy()` est une fonction en **C** qui copie `n` bits depuis une destination vers une source `src`.
Écrivez-en une copie en C#, dont le prototype sera le suivant :

```
static unsafe void *memcpy (void* dst, const void* src, int n);
```

Memset

La fonction `memset()` qui est aussi une fonction en C, remplit les `n` premiers octets de la zone de mémoire pointée par `s` avec l'octet `c`.

Prototype :

```
static unsafe void *memset (void *s, int c, int n);
```

Memcmp

La fonction `memcmp()` compare les `n` premiers octets des zones mémoire `s1` et `s2`. Elle renvoie un entier inférieur, égal, ou supérieur à zero, si `s1` est respectivement inférieure, égale ou supérieure à `s2`.

Prototype :

```
static unsafe int memcmp (const void *s1, const void *s2, int n);
```

Average

La fonction `average()` retourne la moyenne des `size` premières valeurs d'un tableau `src`.

Prototype :

```
static unsafe float average(int* src, int size);
```

Add

La fonction `add()` ajoute `n` au `size` premiers éléments d'un tableau `src`. Elle doit retourner le pointeur vers le premier élément du tableau.

Prototype :

```
static unsafe int* add(int* src, int size, int n);
```

Minimum

La fonction `minimum()` retourne le pointeur vers le plus petit entier contenu dans les `size` premières valeur d'un tableau `src`.

Prototype :

```
static unsafe int* min_val(int* src, int size);
```

Maximum

La fonction `maximum()` retourne l'index du plus grand entier contenu dans les `size` premières valeur d'un tableau `src`.

Prototype :

```
static unsafe int max_val_index(int* src, int size)
```

Sort

La fonction `sort()` doit trier le tableau d'entiers et retourne le pointeur sur son premier élément.
Vous êtes libres quant au choix de l'algorithme de tri (exemple : tri à bulles). Pensez à préciser votre choix dans un commentaire !

Prototype :

```
static unsafe int* sort(int* src, int size);
```

Remove All

La fonction `remove_all()` supprime toutes les occurrences de `n` dans le tableau d'entiers `src` en décalant tout les entiers suivants (à sa droite). La valeur de retour correspond au nombre d'éléments supprimés.
La taille `size` doit être modifiée en conséquence. Prototype :

```
static unsafe int remove_all(int* src, ref int size, int n);
```