Everything you always wanted to know about pointers...

(But were afraid to ask)

1 Submission

Directory tree of your submission :

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

You must, of course, replace `login_x` by your own login.

AUTHORS

Your submission's root must contain an `AUTHORS` file. It lists the project authors' logins as follows : an asterisk , a space, a login (e.g. `login_x`) and a newline.

README

Your submission's root must contain an `README` file. It lists your remarks concerning this TP and your work.

Advice

Don't forget the fundamental rules :

— Your submission's root must contains an `AUTHORS` file ;
— Your submission must contain only what's listed in the directory tree given above, no `bin` or `obj` folder ;
— Your functions must respect the given prototypes ;
— **Your code MUST compile** ;
— If you did some bonuses, don't forget to write it in a `BONI.txt` file.

Remember, if you do not apply these rules, you will face great penalities.

2 Course

The use of pointers is rarely required in C#, but there are some situations that require them. As examples, using an unsafe context to allow pointers is warranted by the following cases :

— Dealing with existing structures on disk ;
— Advanced COM or Platform Invoke scenarios that involve structures with pointers in them ;
— Performance-critical code.

The use of unsafe context in other situations is discouraged. Specifically, an unsafe context should not be used to attempt to write C code in C#.

Example	Description
int* p	p is a pointer to an integer.
int** p	p is a pointer to a pointer to an integer.
int*[] p	p is a single-dimensional array of pointers to integers.
char* p	p is a pointer to a char.
void* p	p is a pointer to an unknown type.

Warning :
The following expression declares three pointers to int :

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

3 Tests on characters

First, in order to use pointers in C#, you must authorise the `unsafe` code context : to do that, go to the Properties of your project, then in `Build` and check `Allow unsafe code`.
You can now use the keyword `unsafe`.

Let's print !

We're going to write a print function with two overloads : one which takes a character and the other a pointer to a character.

Prototypes :

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

Both functions will print the following text in the console :

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

with [1] the character's value and [2] the pointer to it.

Sorry, what ?

Let's add this function :

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

Execute your code... the results are different, why ?
Write a comment above the function to explain what has happened.

Swap

Write a function swap that will swap the values pointed by `p` and `q`.
Prototype :

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

4 Array operations

First, download the `safe.dll` file on `perso.epita.fr`. Once downloaded, place the file into your project folder (see the submission hierarchy) and add it to your project :

— In the Solution Explorer (on the right), right-click on `References` ;
— Choose "Add Reference" ;
— And Browse to the dll.

Now that the library is embedded in your project, add the corresponding `using` at the top of your file.

This will give you access to two classes : `Unsafe` and `Misc`. `Unsafe` contains the corrected version of the functions you must implement, so you will be able to test your TP thoroughly ! Furthermore, you can use `Misc` to generate randomly filled arrays.

Reminder :
A string is simply a array of characters so the operations available for the arrays are also applicable to the string.

Array printing

Write `print_array()` that prints `size` integers of an array `src` followed by a newline.

Prototype :

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

Now, write an overload of this same function that takes only a pointer to character and prints all the characters until the last one, followed by a newline. It must also return the number of characters that have been printed.

Prototype :

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

Advice : The `\0` character marks the end of the string.

Array manipulation

Write `fill_array()` fills the `src` array with the `val` character. It must also return the number of characters that have been modified.

Prototype :

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

Strlen

The `strlen()` function returns an integer corresponding to the length of the array of characters.

Prototype :

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

Clone

The `clone()` function duplicates the array of characters `src` and returns the pointer to its first element.

Prototype :

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

ROTn

Here, you will need to write two `rot()` functions (you should know them perfectly by now !) : they must apply a rotation of `n` characters to the source passed as an argument and return the result of the operation.
The source is a `char` for the first function and a `char*` in the second one.
Advice : Write the first before the second... It can be useful.

Prototype :

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

Remove

The `remove()` function removes the character at `index` and shifts all next chars. The last character will be set to `\0`.

Prototype :

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

Conversion

The `to_string()` function converts an array of characters to a string.

Prototype :

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

Memcpy

The `memcpy()` function is a **C** function which copies `n` bytes from the memory area `src` to memory area `dest`.

Write the C# equivalent with the following prototype :

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

Memset

The `memset()` function fills the first `n` bytes of the memory area pointed to by `s` with the constant byte `c`.

Prototype :

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

Memcmp

The `memcmp()` function returns an integer less than, equal to, or greater than zero if the first `n` bytes of `s1` is found, respectively, to be less than, to match, or be greater than the first n bytes of `s2`.

Prototype :

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

Average

The `average()` function returns the average of the `size` first values of an array `src`.

Prototype :

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

Add

The `add()` function adds `n` to the `size` first elements of an array `src`. It must return the pointer to the first element of the array.

Prototype :

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

Minimum

The `minimum()` function returns the pointer to the smallest integer contained in the `size` first values of an array `src`.

Prototype :

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

Maximum

The `maximum()` function returns the index of the greatest integer contained in the `size` first values of an array `src`.

Prototype :

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

Sort

The `sort()` function must sort the integer array and returns the pointer to its first element. You are free to choose the sorting algorithm you prefer (e.g. Bubble sort), but be sure to specify it in a comment above this function. Prototype :

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

Remove All

The `remove_all()` function removes all occurences of `n` in the integer array `src` by shifting all following integers. The return value is the number of elements that have been removed. The size must be modified accordingly.

Prototype :

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