Erreurs courantes midterm 2023

Accès aux valeurs d'un tableau/pointeur passé en paramètre

```
int NoDupChar(char* A, char* B)
```

A est un **pointeur** sur un char, tableau de char (éventuellement terminé par '\0')
Il est possible d'accéder aux valeurs de A via **A[..]** ou ***(A+..)**
Mais pas via ***A[]**

La fonctionne `strlen(B)` cherche le caractère '\0' indiquant la fin de la chaine de caractère.
Si B ne contient pas le caractère '\0', il y aura un crash une fois que vous essayerez de lire après la taille réservée pour **B**.
Si vous modifiez **B** il faut mettre à jour la position du char '\0' dans **B**.
Ex :

```
char B[10];
B[0] = 'b' ;
B[1] = 'a' ;
B[2] = '\0' ;
int c = strlen(B); // c: 2
```

Calcul avec des pointeurs/tableaux, attention à l'ordre des opérations

```
A[j]   ≠   A+j      ➔ *(A+j)
*(A+1) ≠  *A + 1
```

B[..] retourne un char; les fonctions `fget(),etc.` sont définies pour des fichiers.

Attention aux bornes des boucles et à leur initialisation

```
for (int i; i<= sz; i++) A[sz] ➔ BOOM!  + i pas initialisé!
```

Vous ne pouvez pas modifiez une variable déclarée protégée par **const**

```
int NoDup (const a[]){
      a[2] = 2;  ➔ BOOM!
```

`sizeof(A)/sizeof(A[])` ➔ sizeof(A) on array function parameter **A** will return size of **int ***

```
int SizeArray(int A[]) {
      return sizeof(A)/sizeof(A[0]); // return 2 (fcn. de l'architecture 32/64)
}
```

Taille connue au moment de la compilation -> ok

```
int B[5];
int szB = sizeof(B)/sizeof(B[0]); // ici ok -> szB = 5
```

Assignation dans un test est possible et valide, cependant ce n'est pas une bonne pratique. Le compilateur indique éventuellement un warning.

```
if (i=23) { … }
```

eq.

```
i=23;
if (i) { … } // i ≠ 0 ➔ true
```

En c, il n'est pas possible de retourner plusieurs valeurs via return

```
return U, szU;  // ne compile pas
```

ou

```
return U;     // Ok
return szU;   // jamais éxécuté
```

Il faut passer une des valeurs par pointeur, alternativement vous pouvez retourner une structure avec les valeurs, ou retourné un tableau créé avec `malloc()`, cette dernière option est potentiellement problématique : qui gère le `free()` ?

S'il n'y a pas d'accolades après un if ()/for()/etc. uniquement la ligne suivante (jusqu'au ' ;') est exécutée.

```
if (a>0)
printf("1: %d", a); // sera exécuté seulement si a>0
printf("2: %d", a); // sera toujours exécuté
```

Code inutile

```
if (A != 0) A=0;     ➔    A=0;
```

Opérations bit à bit

```
x & 0 ➔ retourne 0, x=    1011
                        &  0000
                           0000


x & 1 ➔ retourne la valeur du dernier bit de x
                      x=    1011
                        &  0001
                           0001

x & 1 & 1 ➔ retourne la valeur du dernier bit de x
                      x=    1011
                        &  0001
                           0001
                        &  0001
                           0001

x | 1 ➔ force le dernier bit de x à 1
                      x=    1010
                        |  0001
                           1011
```

Les comparaisons en matlab suivent la même syntaxe que le C

```
AllMax = @(V) [max(V), find(V == max(V))]
```