Pour en finir avec les tableaux...

▸ **Exercice 1**. Ecrire une fonction `fact` recevant un paramètre entier `n` et **retournant** l'entier

$$n! = n \times (n-1) \times \cdots \times 2 \times 1$$

Refaire la même fonction, mais cette fois le résultat sera "renvoyé" via un des arguments —la fonction retournera `void`—.

▸ **Exercice 2**. Ecrire deux fonctions `anp` et `cnp` recevant deux paramètres entiers et renvoyant respectivement le nombre d'arrangements

$$A_p^n = \frac{n!}{(n-p)!},$$

et le nombre de combinaisons

$$C_p^n = \frac{n!}{p!(n-p)!}.$$

▸ **Exercice 3**. Ecrire une fonction de calcul et d'affichage du triangle de *Pascal*. Ce fameux triangle de *Pascal* peut se construire de la manière suivante : chaque coefficient d'une ligne est obtenu en additionnant le coefficient situé immédiatement au dessus avec le coefficient situé au dessus à gauche. Les coefficients non définis sont considérés comme étant nuls. On utilisera, dans une première version de la fonction `triangle` deux tableaux `courant` et `precedent` contenant respectivement la ligne en cours de calcul et la ligne précédente. Après calcul et affichage de la ligne courante, on recopiera donc le tableau `courant` dans le tableau `precedent`, etc.

▸ **Exercice 4**. Refaire l'exercice précédent en n'utilisant qu'un seul tableau représentant la ligne en cours de calcul.

Pointeurs et chaînes de caractères...

▸ **Exercice 5**. On considère le programme suivant, et on demande de donner le résultat de son exécution.

```
void main(void)
{
  int t1[]={0, 0, 1, 1, 1, 1}, t2[]={0, 0, 1, 1, 1, 1};
  int *p1, *p2;
  p1=t1;
  p2=t2;
  while(!*p1++ || !*p2++);
  printf("%d %d\n",p1-t1, p2-t2);
}
```

▸ **Exercice 6**. Ecrire une fonction `strcopy()` de copie d'une chaîne de caractères `source` dans une autre chaîne `dest`. La copie s'arrètera lorsque le caractère '\0' sera rencontré. La fonction retournera en plus la chaîne `dest`.

```
char *strcopy(char *dest, const char *source);
```

▸ **Exercice 7**. Ecrire une fonction `strcompare()` de comparaison de deux chaînes de caractères `s1` et `s2` renvoyant 0 si `s1==s2` (au sens de la comparaison des chaînes de caractères...). Si `s1>s2` pour l'ordre lexicographique, on retournera la valeur 1 et sinon la valeur -1.

```
int strcompare(const char *s1, const char *s2);
```

▸ **Exercice 8**. Ecrire un fonction `swap` de swap de deux objets quelconques(1) dont on connait la taille en mémoire. La fonction pourra par exemple retourner le nombre d'octets "swapés".

```
int swapuniversel(char * o1, char * o2, int nboctets);
```

(1) sous-entendu "représentés en mémoire de façon contigüe" !