Exercice 1 : Pointeurs sur tableau de chaînes

Considérez la déclaration suivante, puis pour chaque expression, indiquez le texte affiché sur la sortie standard.

```
char t[][8] = {
    {'A', 'n', 't', 'o', 'i', 'n', 'e', '\0'},
    "Nicolas",
    "Raphael"
};
char *p = t[1];
char (*q)[8] = t;
char **r = &p;
```

(a) `printf("%c", *p);`

```
N
```

(b) `printf("%c", p[5]);`

```
a
```

(c) `printf("%ld", strlen(t[2]));`

```
7
```

(d) `printf("%ld", sizeof(t[0]));`

```
8
```

(e) `printf("%ld", sizeof(t));`

```
24
```

(f) `printf("%c", *(p + 6));`

```
s
```

(g) `printf("%s", p + 3);`

```
olas
```

(h) `printf("%c", (*r)[3]);`

```
o
```

(i) `printf("%s", q[2]);`

```
Raphael
```

(j) `printf("%hhd", (int8_t)t[1][7]);`

```
0
```

Exercice 2 : Pointeurs et tableaux d'entiers

Considérez la déclaration suivante, puis pour chaque expression, indiquez la valeur affichée sur la sortie standard.

```
int b[][3] = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
int *p = b[1];
int (*q)[3] = b;
int **r = &p;
```

(a) `printf("%d", *p);`

> `4`

(b) `printf("%d", p[2]);`

> `6`

(c) `printf("%ld", sizeof(*q));`

> `12`

(d) `printf("%ld", sizeof(b));`

> `36`

(e) `printf("%d", **r);`

> `4`

(f) `printf("%d", q[2][1]);`

> `8`

(g) `printf("%d", *(*q + 1));`

> `2`
> `*q` est équivalent à `q[0]` soit `b[0]`, un `int*` pointant sur `b[0][0]`. `*q + 1` pointe sur `b[0][1]` et la déréférence donne `2`.

(h) `printf("%d", q[p[0] / q[0][2]][2]);`

> `6`
> `p[0]` vaut `4`, `q[0][2]` vaut `3`. La division entière donne `4 / 3 = 1`. L'expression se réduit à `q[1][2]` soit `b[1][2] = 6`.

Exercice 3 : Programmation – strcopy avec void*

La bibliothèque standard C utilise `void*` pour les fonctions génériques de manipulation de mémoire (comme `memcpy`). Un `void*` est un pointeur sans type : il peut recevoir n'importe quel pointeur sans cast explicite, ce qui rend la fonction utilisable pour n'importe quel type de données.

En revanche, on **ne peut pas déréférencer** un `void*` directement : il faut d'abord le caster vers un type concret. Pour traiter des octets un par un, on caste en `char*` :

```
const char *s = (const char *)src;
```

Écrire une fonction qui respecte le prototype ci-dessous. Cette fonction copie une chaîne de caractères (terminée par `'\0'`) depuis `src` vers `dest`.

```
void strcopy(void *dest, const void *src);
```

Contraintes : ne pas utiliser la notation tableau (`a[b]`), utiliser une boucle `while`.

```
void strcopy(void *dest, const void *src)
{
    char *d = (char *)dest;
    const char *s = (const char *)src;
    while (*s)
        *d++ = *s++;
    *d = '\0';
}
```

Exercice 4 : Programmation – réécriture sans variable itérative

On souhaite afficher successivement tous les suffixes d'un tableau d'entiers. Le code suivant, utilisant deux indices, produit la sortie :

```
1 2 3 4 5
2 3 4 5
3 4 5
4 5
5
```

```
int a[] = {1, 2, 3, 4, 5};
int n = 5;

for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++)
        printf("%d ", a[j]);
    printf("\n");
}
```

Réécrire ce programme **sans aucune variable entière itérative** (`i`, `j`, etc.) en faisant progresser des pointeurs et en utilisant la **comparaison de pointeurs** comme condition d'arrêt.

```
int a[] = {1, 2, 3, 4, 5};
int *end = a + 5;

for (int *p = a; p < end; p++) {
    for (int *q = p; q < end; q++)
        printf("%d ", *q);
    printf("\n");
}
```

'end' pointe un élément **au-delà** du dernier élément du tableau (idiome C classique). La boucle externe fait avancer 'p' d'un élément à chaque tour ; la boucle interne parcourt de 'p' jusqu'à 'end' exclu grâce à la comparaison 'q < end'.

Exercice 5 : Programmation – strrev

Écrire une fonction qui respecte le prototype ci-dessous. Cette fonction inverse une chaîne de caractères **sur place** (*in-place*), sans allouer de mémoire supplémentaire.

```
void strrev(char *s);
```

Exemples d'utilisation :

```
char mot[] = "Bonjour";
strrev(mot);   // mot vaut maintenant "ruojnoB"

char vide[] = "";
strrev(vide);  // pas d'effet
```

Contraintes : utiliser deux variables pointeur, ne pas utiliser la notation tableau (`a[b]`).

```
void strrev(char *s)
{
    char *e = s;
    while (*e)
        e++;
    if (e == s) return;
    e--;
    while (s < e) {
        char tmp = *s;
        *s++ = *e;
        *e-- = tmp;
    }
}
```