Solutions: Série 5 - Chaînes de caractères - ICC-C 2025-2026

1. Échauffement : longueur d'une chaîne de caractères

```c
#include <stdio.h>
#include <string.h>

/* Renvoie la longueur de la chaîne `str`, sans compter le \0 final. */
size_t my_strlen(const char str[]) {
  size_t count = 0;
  while (str[count] != '\0') {
    count++;
  }
  return count;
}

int main() {
  printf("%ld\n", my_strlen("foo")); // 3
  printf("%ld\n", my_strlen("Sébastien")); // 10
  printf("%ld\n", my_strlen("👩‍👫")); // 26

  return 0;
}
```

Le résultat 10 pour `"Sébastien"` vient du fait que é est un « caractère » (un *code point* pour être exact) qui est encodé sur 2 octets, et donc 2 `char`. Il contribue donc pour 2 dans le calcul de la longueur.

L'émoji 👩‍👫 est encore plus bizarre. Il est en fait formé de 6 code points, chacun encodé sur plusieurs `char`, pour un total de 26 `char`.

Ce que vous devez retenir de cela, c'est qu'un seul `char` ne correspond pas à notre notion visuelle de « caractère ». Ce n'est une approximation valable que pour les lettres latines non accentuées, les chiffres arabes et quelques signes de ponctuation utilisés en langue anglaise.

2. Occurrences

```c
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <assert.h>

/* Teste si on trouve la sous-chaîne `substr` dans `str` à l'indice `p`. */
bool a_sous_chaine(const char substr[], const char str[], size_t p) {
  size_t i = 0;
  while (substr[i] != '\0') {
    if (str[p + i] != substr[i]) {
      /* On n'a pas encore atteint la fin de `substr`, mais un caractère est
       * différent dans `str`.
       * Subtilité : quand on arrive au bout de `str` elle-même, on y trouvera
       * un `str[p + i] == '\0'. Puisqu'on sait que `substr[i] != '\0' quand
       * on arrive au test, on entrera dans cette branche-ci aussi.
       * C'est grâce à cela qu'on ne va pas dépasser la fin de `str` !
       */
      return false;
    }
    i++;
  }

  /* On a atteint la fin de substr et tous les caractères correspondaient
   * dans str.
   */
  return true;
}

/* Compte le nombre d'occurrences de `substr` dans `str`. */
int occurrences(const char substr[], const char str[]) {
  int result = 0;
  size_t p = 0;
  while (str[p] != '\0') {
    if (a_sous_chaine(substr, str, p)) {
      result++;
    }
    p++;
  }
  return result;
}

void remove_trailing_newline(char str[]) {
  size_t len = strlen(str);
  assert(len > 0);
  str[len - 1] = '\0';
}
```

```c
int main() {
  const size_t max_line_size = 256 + 1; // + 1 pour le \0 final

  char substr[max_line_size];
  fgets(substr, max_line_size, stdin);

  // Subtilité : il faut retirer le \n qui a été stocké par fgets.
  remove_trailing_newline(substr);

  char str[max_line_size];
  fgets(str, max_line_size, stdin);
  remove_trailing_newline(str); // moins nécessaire dans ce cas-ci

  int nombre_occurrences = occurrences(substr, str);
  printf("%d\n", nombre_occurrences);

  return 0;
}
```

3. La menace des reines

```c
#include <stdio.h>
#include <stdbool.h>

size_t read_int() {
  int x;
  scanf("%d", &x);
  return x;
}

/* Teste si la reine en (queen_row, queen_col) attaque la position
 * (pos_row, pos_col).
 */
bool is_attacking(int pos_row, int pos_col, int queen_row, int queen_col) {
  return (
    pos_row == queen_row // même ligne
    || pos_col == queen_col // même colonne
    || (pos_row - queen_row) == (pos_col - queen_col) // même diagonale 1
    || (pos_row - queen_row) == (queen_col - pos_col) // même diagonale 2
  );
}

/* Parmi les reines en positions (queen_rows[i], queen_cols[i]) pour tout
 * 0 <= i < k, compte le nombre d'entre elles qui attaquent la position
 * (pos_row, pos_col).
 */
int number_of_attacking_queens(int pos_row, int pos_col, int queens_rows[],
    int queens_cols[], size_t k) {

  int result = 0;
  for (size_t i = 0; i < k; i++) {
    if (is_attacking(pos_row, pos_col, queens_rows[i], queens_cols[i])) {
      result++;
    }
  }
  return result;
}

int main() {
  int pos_row = read_int();
  int pos_col = read_int();

  size_t k = (size_t) read_int();
  int queens_rows[k];
  int queens_cols[k];
  for (size_t i = 0; i < k; i++) {
    queens_rows[i] = read_int();
    queens_cols[i] = read_int();
  }

  int total = number_of_attacking_queens(pos_row, pos_col, queens_rows,
      queens_cols, k);
  printf("%d\n", total);

  return 0;
}
```