Identification

Prénom: .....

Nom: .....

Code permanent: .....

Veuillez choisir **une seule réponse**, avec une croix (×) ou un crochet (✓).
Considérez le programme troué suivant:

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

typedef struct {
  unsigned int length;
  char* string;
} String;

void string_initialize(String* s1,
                       const char* s2) {
  s1->length = strlen(s2);
  s1->string = strdup(s2);
}

void string_delete(String* s) {
  free(/* TODO 1 */);
}

void string_concatenate(String* s1,
                        const String* s2) {
  s1->length += s2->length;
  s1->string = realloc(/* TODO 2 */, /* TODO 3 */);
  strcat(/* TODO 4 */, /* TODO 5 */);
}

int main(void) {
  String s1, s2, s3;
  string_initialize(&s1, "");
  string_initialize(&s2, "abc");
  string_initialize(&s3, "de");
  printf("\"%s\"\n", s1.string);
  string_concatenate(&s1, &s2);
  printf("\"%s\"\n", s1.string);
  string_concatenate(&s1, &s3);
  printf("\"%s\"\n", s1.string);
  string_delete(&s1);
  string_delete(&s2);
  string_delete(&s3);
  return 0;
}
```

Q1. Que devrait-on mettre à la place de `/* TODO 1 */`?

- ☐ `*s`
- ☐ `&s->string`
- ☐ `s`
- ■ `s->string`

Q2. Que devrait-on mettre à la place de `/* TODO 2 */`?

- ☐ `s1`
- ☐ `*s1->string`
- ☐ `&s1->string`
- ■ `s1->string`

Q3. Que devrait-on mettre à la place de `/* TODO 3 */`?

- ■ `(s1->length + 1) * sizeof(char)`
- ☐ `(s1->length + 1) * sizeof(char*)`
- ☐ `s1->length * sizeof(char)`
- ☐ `s1->length * sizeof(char*)`

Q4. Que devrait-on mettre à la place de `/* TODO 4 */`?

- ☐ `&s1->string`
- ☐ `&s2->string`
- ■ `s1->string`
- ☐ `s2->string`

Q5. Que devrait-on mettre à la place de `/* TODO 5 */`?

- ☐ `&s1->string`
- ☐ `&s2->string`
- ☐ `s1->string`
- ■ `s2->string`

Q6. Quel nom donne-t-on à la partie publique d'un module?

- ☐ définition
- ☐ implémentation
- ☐ documentation
- ■ interface

Q7. En C, quel adjectif utilise-t-on pour désigner une fonction qui peut avoir un nombre variable d'arguments?

- ☐ arbitraire
- ☐ macro
- ☐ flexible
- ■ variadique

Q8. Qu'est-ce qui sera affiché sur la sortie standard si on exécute le programme suivant?

```
#include <stdio.h>

#define abs(X) X >= 0 ? X : -X

int main(void) {
  int x = -3;
  printf("%d\n", abs(x - 4));
  return 0;
}
```

- ■ `-1`
- ☐ `1`
- ☐ `-7`
- ☐ `7`

Q1. Dans `string_initialize`, la fonction `strdup` réserve de la mémoire pour stocker la chaîne de caractères `s1->string`. Il faut donc libérer cet espace mémoire dans `string_delete` avec `free(s->string)`.

Q2. Lors d'une réallocation de mémoire, on passe le pointeur qui pointe vers le bloc de mémoire qui doit être réalloué, dans le cas présent, `s1->string`. Noter qu'une réallocation peut exiger une relocalisation de l'adresse, de sorte qu'il est important de stocker la valeur de retour de `realloc` dans `s1->string`.

Q3. Le bloc mémoire qui doit être réservé lors de la réallocation doit pouvoir stocker la nouvelle chaîne de caractères concaténée, c'est-à-dire autant de caractères, plus 1, en raison du caractères `\0`, pour que la chaîne soit bien formée. On réserve donc `s1->length + 1` cases de taille individuelle `sizeof(char)`.

Q4 et **Q5**. Lors de la concaténation, la chaîne *destination* est `s1->string` (premier argument de `strcat`) et la chaîne *source* est `s2->string` (deuxième argument de `strcat`).

Q6. La partie publique d'un module est appelée *interface*. Le mot *définition* est employé en C pour désigner le contenu du corps d'une fonction, par exemple, par opposition à *déclaration*, qui désigne plutôt la signature de la fonction. La notion de *documentation* n'est pas réservée à la partie publique: on peut très bien documenter la partie privée d'un module, cette documentation s'adressant alors plutôt aux personnes responsables du développement ou de la maintenance de ce module. Finalement, le mot *implémentation* n'est pas non plus spécifique aux notions de visibilité publique ou privée: on trouve généralement l'implémentation d'un module dans sa partie privée, mais on pourrait aussi, dans certains cas, la retrouver dans sa partie publique.

Q7. Une fonction ayant un nombre variable d'arguments est appelée fonction *variadique*. En C, Le mot *macro* désigne plutôt une directive déclarée à l'aide de `#define`. On distingue deux types de macro: une macro simple, qui déclare un symbole, ainsi qu'une macro-fonction, qui permet de faire de la métaprogrammation. Noter qu'une macro-fonction peut être variadique.

Q8. Cette question met en évidence les risques associés aux macro-fonctions. Comme la définition de `abs(X)` n'utilise pas de parenthèses, lors du prétraitement, l'expression `abs(x - 4)` sera remplacée par `x - 4 >= 0 ? x - 4 : -x - 4`. Or, on a que `x - 4 == -3 - 4 == -7`, de sorte que `x - 4 >= 0` est fausse, ce qui entraîne que la valeur retournée sera `-x - 4 == --3 - 4 == 3 - 4 == -1`.