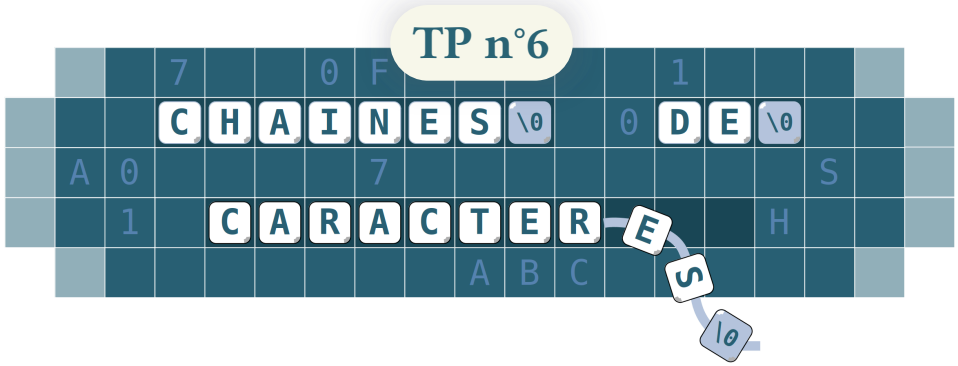



## I. Chaînes de caractères et mémoire

**1** Représenter l'organisation mémoire de ce processus. Indiquer où est stocké `s2`. Indiquer où sont `s1`, `s3` et `s4` et où ils pointent. Quel problème grave voyez-vous ?

```c
char* bonjour() {
  char s[] = "bonjour !";
  return s;
}

char* bonjour2() {
  return "bonjour !";
}

int main() {

  char* s1 = "bonjour !";
  char s2[] = "bonjour !";
  char* s3 = bonjour();
  char* s4 = bonjour2();

  return 0;
}
```

**2** On souhaite modifier la valeur d'une lettre. Que se passe-t-il si l'on écrit `s1[4] = 'a'` ? Pourquoi cela ?
Même question pour `s2`, `s3` et `s4`.

## II. Allocation dynamique

**3** Si l'on souhaite allouer dynamiquement une zone mémoire dans le tas afin d'y stocker une chaîne de $n$ caractères, quelle fonction doit-on appeler, et que doit-on lui donner comme argument ? Que peut-on dire des valeurs initialement présentes dans cet espace alloué ?

**4** Quels avantages et inconvénients voyez-vous par rapport aux méthodes précédentes ?



## III. Manipulations de chaînes de caractères

### Question filée

Dans cette partie, il est demandé d'annoter **toutes** vos fonctions avec leur complexité temporelle pire cas.

**5** Dans un fichier **chaines.c**, implémenter une fonction de prototype `int strlen(char *s)` renvoyant la longueur d'une chaîne de caractères valide `s` donnée. Par exemple, `strlen("Hello")` renvoie 5.

**6** Implémenter une fonction de prototype `int nb_occurrences(char *s, char c)` renvoyant le nombre d'occurrences du caractère `'c'` dans la chaîne de caractères `s`.

**7** Implémenter une fonction de prototype `char *cri(int l)` qui alloue et renvoie une chaîne de caractères valide de longueur `l` et ne contenant que des `'A'`. Par exemple, `cri(7)` doit allouer et renvoyer une chaîne de valeur `"AAAAAAA"`. Attention au caractère sentinelle `'\0'` qu'il ne faut pas oublier de placer en fin de chaîne afin que celle-ci soit valide.

**8** Que se passe-t-il si l'on appelle la fonction `strlen` sur une chaîne de caractères invalide qui n'aurait pas de caractère sentinelle à la fin ? Le programme compilera-t-il ? Quelle erreur risque-t-on de rencontrer ?

**9** Implémenter une fonction de prototype `char *strcpy(char *dest, char *source)` qui recopie une chaîne de caractères `source` dans une zone allouée pointée par `dest`, puis qui renvoie le pointeur vers `dest`.

**10** Implémenter une fonction de prototype `char *strcat(char *s1, char *s2)` qui alloue et renvoie une nouvelle chaîne de caractères dont la valeur est la concaténation de `s1` et `s2`. Par exemple, `strcat("Val", "bonne")` doit allouer et renvoyer une chaîne de caractères de valeur `"Valbonne"`.

**11** Implémenter une fonction de prototype `int strcmp(char *s1, char *s2)`, qui :
- renvoie 0 si les deux chaînes sont de même valeur,
- renvoie −1 si `s1` apparaît **avant** `s2` dans l'ordre du dictionnaire français,
- renvoie 1 si `s1` apparaît **après** `s2` dans l'ordre du dictionnaire français.

### Ces fonctions existent déjà !

Les fonctions `strlen`, `strcpy`, `strcat` et `strcmp` sont déjà présentes dans la librairie C. Maintenant que vous les avez implémentées et que vous connaissez leurs complexités, vous n'avez plus besoin de les réécrire à chaque fois. Il vous suffira d'importer la librairie **string** en amont pour pouvoir les invoquer librement dans vos TP.

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

**12** Dans un terminal, lire la documentation de ces quatre fonctions avec la commande **man**. Remarquer la différence entre votre fonction `strcat` et celle de la librairie.

```
man strlen
```

## IV. Tests

**13** Implémenter une fonction de tests permettant de vérifier le bon comportement de vos implémentions de la partie précédente sur différents exemples pertinents. Compiler et lancer vos tests.

Comme votre programme réalise des allocations dynamiques de mémoire, il faut également s'assurer que **tout ce qui a été alloué a bien été libéré**. Pour cela, un **code sanitizer** (ou simplement appelé **sanitizer**) peut aider à détecter des comportements problématiques, en générant un exécutable plus lent mais qui vérifie la salubrité de la mémoire à la volée. Avec le compilateur gcc, il est possible d'intégrer un code sanitizer de la manière suivante :

```
gcc -o chaines chaines.c -fsanitize=address,undefined
```

puis

```
./chaines
```

**14** Le sanitizer vous affiche-t-il des problèmes liés à la gestion de la mémoire ? Si c'est le cas, d'où viennent-ils et comment les résoudre ? **Il est important de ne pas passer à la suite du TP tant que tout n'est pas parfaitement réglé, quitte à y passer un peu de temps.**

## V. Recherche naïve d'un motif dans un texte

**15** Pour $t_0 t_1 ... t_{n-1}$ une chaîne de $n$ caractères, on appelle facteurs de $t$ les chaînes $t_i t_{i+1} ... t_j$ pour $0 \leq i \leq j \leq n-1$, ainsi que la chaîne vide. Combien de facteurs possède une chaîne de caractères de longueur $n$ ? Et de facteurs de longueur exactement $k$ ?

**16** Implémenter une fonction de prototype `char *facteur(char *s, int debut, int fin)` qui alloue et renvoie une chaîne de caractères correspondant au facteur de `s` compris entre l'indice `debut` (inclus) et `fin` (inclus). Par exemple, `facteur("Valbonne", 2, 5)` doit renvoyer `"lbon"`.

On cherche à implémenter une fonction de prototype `bool est_facteur(char *texte, char *motif)` qui vérifie si `motif` est un des facteurs de la chaîne `texte` ou pas. Voici une tentative erronée :

```c
bool est_facteur(char *texte, char *motif) {
  int n = strlen(texte);
  int k = strlen(motif);
  for (int i=0; i<n-k+1; i++) {
    char *fact = facteur(texte, i, i+k-1);
    if (fact == motif) {
      return true;
    }
  }
  return false;
}
```

**17** Pourquoi cette fonction n'est pas correcte ? Que risque-t-il de se passer ? Que pensez-vous de la gestion mémoire de cette fonction ? Proposer et implémenter une version sans ces problèmes.

**18** Quelle est la complexité temporelle pire cas de la fonction ainsi obtenue ?



## VI. Opérations sur les caractères

**19** Implémenter une fonction de prototype `int nombre_majuscules(char *s)` qui renvoie le nombre de lettres majuscules présentes dans la chaîne `s`.

**20** Quelle est la valeur de la variable `char mystere = 't' - 'a' + 'A';` ?

**21** Implémenter une fonction de prototype `char *en_majuscules(char *s)` qui alloue et renvoie une nouvelle chaîne de même valeur que `s`, à l'exception des lettres minuscules qui deviennent des lettres majuscules. Par exemple `en_majuscules("Bonjour !")` doit allouer et renvoyer une chaîne de valeur `"BONJOUR !"`.

**22** Dans un nouveau programme C, écrire une fonction `char *cesar(char *s, int d)` qui alloue et renvoie une nouvelle chaîne de caractères dans laquelle les lettres de `s` ont été décalées circulairement de `d` positions dans l'ordre alphabétique. Par exemple avec `d=3`, `'a'` devient `'d'`, `'b'` devient `'e'`, ... `'y'` devient `'b'` et `'z'` devient `'c'`. Les autres caractères restent inchangés.

**23** Modifier la fonction `main` afin que le programme puisse prendre en entrée un entier et une chaîne de caractères, et affiche ensuite le chiffrement de César de cette chaîne avec le décalage donné.

## VII. Un très mauvais caractère...

Dans le jeu-vidéo Zelda Ocarina of Time, un marathonien malhonnête vous nargue en affirmant toujours terminer la course une seconde avant vous. Si (grâce à quelques astuces) vous parvenez à obtenir un temps de 0 seconde, il persistera alors dans son mensonge ce qui provoquera un un affichage intrigant de 0 minute et `/` seconde.




**24** Dans un nouveau programme C, reproduire ce bug d'affichage pour expliquer le plus précisément possible sa cause, et détailler comment on aurait pu l'éviter.