Programmation C
TP n° 6 : Chaînes de caractères

Dans cet énoncé :

- vous pouvez utiliser les fonctions de la bibliothèque standard vues en cours, notamment celles définies dans `ctype.h` et `string.h` ;
- à chaque fois que vous faites un `malloc`, assurez-vous avec `assert` que la mémoire a été correctement allouée ;
- veillez à bien tester chacune des fonctions.

1 Échauffement

Exercice 1 :

Quels comportements ont les fonctions suivantes et que calculent/affichent elles ? Attention, certaines comportent des erreurs à ne pas commettre !

1.

```c
void f1 (const char *s) {
  for(int i = 0; s[i] != '\0'; i++)
    printf ("%c", s[i]);
}
```

```c
void f1 (const char *s) {
  for(; *s; s++)
    printf("%c", *s);
}
```

2.

```c
char *g1 (const char *s) {
  char *t;
  strcpy (t, s);
  return t;
}
```

```c
char *g2 (const char *s) {
  char *t = malloc (strlen (s));
  strcpy (t, s);
  return t;
}
```

```c
char *g3 (const char *s) {
  char *t = malloc (strlen (s) + 1);
  strcpy (t, s);
  return t;
}
```

2 Chaînes et mots

Dans les exercices qui suivent, par *mot* d'une chaîne, on entend toute suite non vide maximale de caractères adjacents dans la chaîne et tous différents des caractères d'espacement – `' '`, `'\n'`, `'\t'`, ... et plus généralement tous les caractères pour lesquels la fonction `int isspace (int c)` renvoie une valeur non nulle [1].

1. Le type du paramètre de `isspace` est `int` et non `char`, afin de permettre l'application de cette fonction à la constante `EOF` ("end of file") qui est la valeur renvoyée par certaines fonctions d'entrées-sorties pour signaler la fin d'un flux de caractères.

Par exemple, la suite des mots de `" a aa ba a bbbb "` est `"a"`, `"aa"`, `"ba"`, `"a"` et `"bbbb"`. Les remarques suivantes sont à garder en tête dans les exercices qui suivent :

- Il peut y avoir plusieurs espacements entre deux mots.
- Il peut y avoir des espacements avant le premier ou après le dernier, mais ce n'est pas obligatoire. Le début du premier mot peut être le premier caractère de la chaîne. La fin du dernier mot peut être le dernier caractère de la chaîne.
- Même non vide, une chaîne peut ne contenir aucun mot – si elle ne contient que des caractères d'espacement.

Dans les exercices qui suivent, rappelez-vous que le dernier mot d'une chaîne n'est donc pas toujours suivi d'un espacement : il peut être suivi du caractère nul.

Exercice 2 : Nombre de mots

Écrire une fonction `int nbr_words(const char *s)` renvoyant le nombre de mots de la chaîne d'adresse `s`. Par exemple, si cette chaîne est `" a aa ba a bbbb "`, la fonction doit renvoyer `5`. Servez-vous de `isspace`.

Exercice 3 : Extraction de mots

1. Écrire une fonction `int word_len(char *w)`. Cette fonction suppose que `w` est l'adresse d'un départ de mot dans une chaîne de caractères. Elle doit renvoyer la longueur de ce mot. Par exemple, si la chaîne `" abc d"` est à l'adresse `s`, l'adresse `s + 1` est celle du caractère `'a'` dans la chaîne, et `word_len(s + 1)` renvoie `3`.
2. Écrire une fonction `char *extract_word(const char *w, int *pl)`. Cette fonction suppose aussi que `w` est l'adresse d'un départ de mot dans une chaîne de caractères. Elle doit effectuer le traitement suivant :
   1. Allouer un zone mémoire permettant de stocker une copie de ce mot sous forme de chaîne de caractères (servez-vous de `word_len`).
   2. Copier le mot dans la zone allouée (par une boucle – la prochaine fois, servez-vous de `strncpy`, mais gardez en tête qu'un appel de `strcpy` ou de `strncpy` n'est jamais gratuit).
   3. Inscrire la longueur du mot à l'adresse `pl`.
   4. Renvoyer l'adresse d'allocation.

3 Indexs

Dans cette partie, nous allons définir un type de structure permettant de manipuler des zones-mémoire allouées dynamiquement et contenant des adresses de chaînes sans espacements. Ce type est :

```
1  typedef struct {
2    int nbr;
3    char **words;
4  } w_index;
```

Nous appelerons *index* toute valeur de type `w_index` respectant les contraintes suivantes :

1. La structure a été allouée par `malloc`.
2. La valeur de son champ `words` est l'adresse d'une zone mémoire allouée par `malloc` contenant une suite de `nbr` pointeurs de type `char *`.
3. Chacun de ces pointeurs est l'adresse d'une chaîne non vide sans espacements, c'est-à-dire réduite à un mot, toujours allouée par `malloc`.

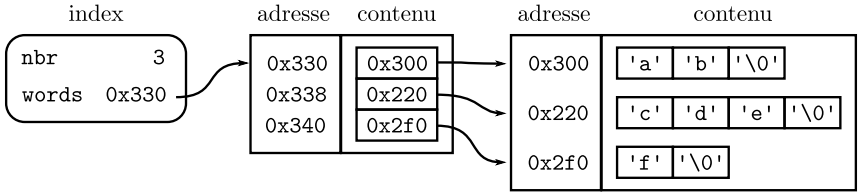

FIGURE 1 – Exemple d'index.

La Figure 1 est un exemple d'index (les adresses sont fictives) contenant les trois mots `"ab"`, `"cde"` et `"f"`. Les adresses de ces mots (`0x300`, `0x220`, `2f0`) sont stockées à l'adresse `0x330`, qui est aussi la valeur du champ `words` de l'index.

Exercice 4 :

En supposant que `pi` est l'adresse d'un index (avec les contraintes sur les allocations-mémoire), écrire `void free_index(w_index *pi)` libérant par des `free` *tout* l'espace-mémoire alloué pour cet index.

Exercice 5 : Concaténation

1. Écrire une fonction `int size_words(w_index *pi)`. Cette fonction suppose que `pi` est l'adresse d'un index. Elle doit renvoyer le nombre total de caractères des mots de l'index.
2. Écrire une fonction `char *concat_words(w_index *pi)`. Cette fonction doit allouer par `malloc`, construire et renvoyer l'adresse d'une nouvelle chaîne formée de la concaténation de tous les mots de l'index, en séparant chaque couple de mots successifs par un unique espace. Servez-vous bien sûr de `size_words`, et n'oubliez pas de tenir compte des espaces et du caractère nul dans le calcul de la taille d'allocation.
   Remarque. Rien n'interdit dans la définition d'un index que son ensemble de mots soit vide (`nbr` vaut `0`). Dans ce cas particulier, la fonction doit renvoyer une chaîne vide.

Exercice 6 : Construction d'un index

Écrire une fonction `w_index *cons_index(const char *s)`. Cette fonction suppose que `s` est l'adresse d'une chaîne de caractères. Elle doit allouer par `malloc` une structure de type `w_index`, l'initialiser pour former l'index des mots de la chaîne, puis renvoyer l'adresse de l'index.

Il faudra donc initialiser la valeur du champ `words` de la structure par un second `malloc` en allouant une zone-mémoire permettant de stocker `nbr_words(s)` pointeurs vers `char`.

Soit `pi` l'adresse de la structure allouée. La valeur de `nbr_words` sera copiée dans `pi -> nbr`. Pour chaque mot de la chaîne de rang `i` dans la suite des mots de la chaîne, la fonction doit initialiser `pi -> words[i]` à l'adresse d'une copie de ce mot – servez-vous de la fonction `extract_word` de l'Exercice 3, et de la longueur du mot inscrite par cette fonction dans une variable pour vous décaler dans la chaîne après le dernier mot recopié.

Exemple. Considérons le code suivant :

```
1  w_index *pi = cons_index(" ab   cde  f ");
2  char *s = concat_words (pi);
3  printf ("%s\n", s);
4  free (s);
5  free_index (pi);
```

Après l'appel de `cons_index`, le pointeur `pi` vaut l'adresse d'une nouvelle structure de type `w_index`, et `pi -> words` contient l'adresse d'une zone-mémoire contenant trois adresses de chaînes :

- `p -> nbr` vaut `3`,
- `p -> words[0]` est l'adresse d'une nouvelle chaîne `"ab"`,
- `p -> words[1]` est l'adresse d'une nouvelle chaîne `"cde"`,
- `p -> words[2]` est l'adresse d'une nouvelle chaîne `"f"`.

En appliquant `concat_words` à `pi`, on obtient une chaîne contenant la même suite de mots que la chaîne initiale - sans espaces avant le premier mot et après le dernier, et avec un seul espace entre chaque couple de mots adjacents.