TP6 – Structures et pointeurs

Langage C (LC4)

Semaine du 11 mars 2013

1 Chaînes de caractères

On va utiliser les fonctions de la bibliothèque standard destinées à la manipulation des chaînes de caractères. Il ne faut pas oublier d'inclure `<string.h>` avant de les utiliser.
On se donne la structure suivante :

```
struct traduction {char *a; char *b; };
```

On peut alors représenter un dictionnaire bilingue par un tableau de `struct traduction`, chaque case du tableau contenant une paire de mots se traduisant l'un en l'autre. On supposera que les dictionnaires sont triés par ordre lexicographique (l'ordre alphabétique) sur le champ `a`.

Question 1. Écrire une fonction `char *recherche(struct traduction *dico, int n, char *s)` qui recherche le mot `s` dans les champs `a` des `struct traduction` du dictionnaire `dico` (de taille `n`), et renvoie en résultat sa traduction, ou `NULL` si elle ne trouve pas `s`.
Comme le dictionnaire est supposé trié, utilisez une recherche dichotomique, ce qui est nettement plus efficace qu'une recherche linéaire.
Utilisez la fonction `int strcmp(char *s, char *t)` pour comparer deux chaînes `s` et `t` : elle renvoie un nombre négatif si la chaîne `s` est avant `t` (pour l'ordre lexicographique), 0 si elles sont égales, et un nombre positif sinon.

Question 2. Écrire une fonction `int nombre_espaces(char* s)` qui renvoie le nombre de caractères « espace » présents dans la chaîne `s`.

Question 3. En utilisant la fonction précédente, écrire une fonction `char **decoupe(char *s)` qui découpe la chaîne `s` en mots, en fait un tableau de `char*` dont le dernier élément est égal à `NULL`.
Par exemple, si `s` vaut `"ga bu zo meu"`, la fonction `decoupe` devra renvoyer le tableau : `{"ga", "bu", "zo", "meu", NULL}`.

Vous pourrez utiliser :
- la fonction `char *strchr(const char *s, int c)` qui renvoie l'adresse de la première occurrence du caractère `c` dans la chaîne `s` en partant du début de la chaîne.
- la fonction `char *strcpy(char *dst, const char *src)` qui copie la chaîne `src` dans `dst`, y compris le caractère de fin de chaîne (et qui renvoie `dst`).
  Attention : `strcpy` n'alloue pas de mémoire, il faut donc que `dst` désigne une chaîne de caractère suffisamment longue,
- la fonction `char *strncpy(char *dst, const char *src, size_t n)`, identique sauf que pas plus de n caractères de `src` ne seront copiés (donc, s'il n'y a pas de caractère nul dans les n premiers caractères de `src`, le résultat n'aura pas de caractère de fin de chaîne).

Question 4. Écrire une fonction `char *reconstruit(char **tab)` qui effectue la transformation inverse.
Vous pourrez utiliser :
- `size_t strlen(const char *s)` qui renvoie le nombre de caractères de s sans tenir compte du caractère de fin de chaîne,
- `char *strcat(char *dst, const char *s)` qui concatène la chaîne `s` à la suite de `dst` (et renvoie `dst`).
  Attention : même remarque que pour `strcpy`, il faut que `dst` soit suffisamment grand. `strcat` écrit par dessus le caractère `'\0'` à la fin de `dst` puis ajoute un `'\0'` à la fin de la concaténation.

Question 5. À l'aide des fonctions précédentes, écrire une fonction `char *traducteur(struct traduction *dictionnaire, int n, char *s)` qui traduit mot à mot la chaîne `s` à l'aide du dictionnaire `dico` contenant `n` mots.

2 Mémoire

En incluant `<string.h>`, on a aussi accès à des fonctions destinées à la manipulation de la mémoire. La fonction `void *memcpy(void *dest, const void *src, size_t n)` permet de copier `n` octets de la zone mémoire pointée par `src` vers la zone mémoire pointée par