TP6 – Structures et pointeurs

Langage C (LC4)

Semaine du 12 mars 2012

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.

► **Exercice 1**

```
char *recherche(struct traduction *dico, int n, char *s)
{
  int i = 0, j = n, k, comp;
  while (i < j) {
    k=(i+j)/2;
    comp=strcmp(dico[k].a, s);
    if (comp < 0)
      i=k;
    if (comp > 0)
      j=k;
    if (comp == 0)
      return dico[k].b;
  }
  return NULL;
}
```

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

► **Exercice 2**

```
int nombre_espaces(char *s) {
  int compte = 0;
  while (*s) {
    if (*s == ' ')
      compte++;
    s++;
```

```
  }
  return compte;
}
```

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).

► **Exercice 3**

```
char **decoupe(char *s)
{
  int cpt = nombre_espaces(s);
  char *t = s;
  char **res;
  res=malloc((cpt + 1) * sizeof(char*));
  cpt = 0;
  t = s;
  s = strchr(t, ' ');
  while (s != NULL) {
    res[cpt] = malloc((s - t + 1));
    strncpy(res[cpt], t, s-t);
    res[cpt][s - t] = '\0';
    s++;
    t = s;
    s = strchr(t, ' ');
    cpt++;
  }
  res[cpt] = malloc(strlen(t) + 1);
  strcpy(res[cpt], t);
  res[cpt + 1] = NULL;
  return res;
}
```

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.

► **Exercice 4**

```
char *reconstruit(char **tab){
  char *res;
  int len = 0;
```

```
  int i = 0;
  while (tab[i] != NULL){
    len += strlen(tab[i])+1;
    i++;
  }
  res = malloc(len);
  *res = '\0';
  i = 0;
  while (tab[i] != NULL){
    strcat(res, tab[i]);
    strcat(res, " ");
    i++;
  }
  return res;
}
```

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.

► **Exercice 5**

```
char *traducteur(struct traduction *dico, int n, char *s){
  char **dec=decoupe(s);
  char **d = dec;
  while (*d != NULL){
    *d = recherche(dico,n,*d);
    if (*d == NULL)
    *d = "?";
    d++;
  }
  return(reconstruit(dec));
}
```

2 Polynômes

On représente toujours un polynôme par un tableau de double. Mais on regroupe le pointeur vers les coefficients et le degré du polynôme dans une structure (Rappel : un polynôme de degré d a $d+1$ coefficients) :

```
typedef struct { int degre; double *coefficients; } polynome;
```

Question 6. Écrire une fonction `double valeur_polynome(int n, polynome *P)` qui évalue la valeur du polynôme P en n.

Question 7. Écrire une fonction `polynome *somme_polynome (polynome *P, polynome *Q)` qui calcule la somme des polynômes P et Q.

Question 8. Écrire une fonction `polynome *produit_polynome (polynome* P, polynome* Q)` qui calcule le produit des polynômes P et Q.

Question 9. Écrire une fonction `polynome *genere_polynomes (double *t, int tsize)` qui étant donné un tableau `t` contenant `tsize` doubles renvoie un tableau de polynomes contenant à la case i, le polynome $P_i(X) = X - t[i]$.

Question 10. Écrire une fonction `polynome* multi_produit_polynome(polynome* P, int pcount)` qui étant donné un tableau de `pcount` polynomes calcule leur produit.