TD 7 : Fonctions et chaînes de caractères

Les chaînes de caractères sont stockées dans des tableaux en terminant la suite par `'\0'`

```
char S1[100], S2[100] ;    /* déclaration de 2 chaînes */
fgets(S1, taille, stdin) ; /* lecture d'une chaîne */
printf("%s", S1) ;         /* écriture d'une chaîne */
strcmp(S1, S2) ;  /* retourne -1, 0 ou 1 suivant que S1<, = ou >S2)*/
strcpy(S1, S2) ;  /* recopie S2 dans S1
```

1°) Programmer les fonctions booléennes (qui retournent 0 ou 1)
- prefixe(a, b) qui vérifie si a est une sous chaîne qui débute la chaîne de caractères b.
- sousChaine(a, b) qui vérifie si a est une sous chaîne de la chaîne de caractères b.

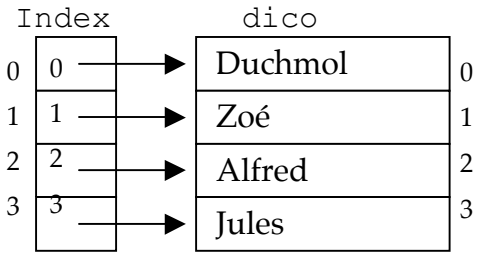
2°) On souhaite trier (tri par permutation) un tableau `dico` de n chaînes de caractères.
- Déclarez un tableau de 50 chaînes de 20 caractères au plus,
- Ecrivez la fonction qui trie un tableau de ce type passé en paramètre,

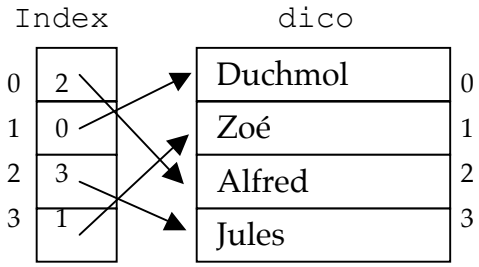
3°) On souhaite faire le tri précédent en minimisant les manipulations (la recopie d'une chaîne coûte « cher »). Nous allons utiliser pour cela une « indirection » :
Soit un tableau `dico[n]` de n chaînes et un tableau de n entiers `Index[n]` (*attention,* `"index"` *,sans majuscule, est un mot réservé du C*). On accède à une chaîne i par `dico[Index[i]]` et on ordonne le tableau Index de sorte que son parcours séquentiel désigne les chaînes dans l'ordre alphabétique.

Soit le main suivant :

```
int main(void)
{ lireDico() ;
  triVirtuelDico() ;
  afficherDico() ;
  return 0 ;
}
```

Ecrire les trois fonctions qui mettent en place cette indirection (*lireDico*) et l'utilisent (*triVirtuelDico* et *afficherDico*).





Remarque : le tableau `Index` aurait pu contenir directement les adresses des chaînes plutôt que leurs indices dans `dico`. Les chaînes de caractères auraient aussi pu être allouées au fur et à mesure des besoins.

Correction TD 7 : Fonctions et chaînes de caractères

1°) solutions non optimisees mais simples
En décomposant :

```
int prefixe((char a[], char b[])
/* b commence par a ? */
{ int i ;
  while (a[i] != '\0')
  { if (a[i] != b[i] ) return 0 ;
    i++ ;
  }
  return 1 ;
}

int sousChaine(char a[], char b[])
/* b contient a ? */
{ int i ;
  i = 0 ;
  while (b[i] != '\0')
  { if ( prefixe(a, &(b[i])) ) return 1 ;
    /* ou encore if ( prefixe(a, b+i)) ) return 1 ; */
    i++ ;
  }
  return 0 ;
}
```

Sans décomposer :

```
int sousChaineBis(char a[], char b[])
{ int i, j, k;
  i = 0 ;
  while (b[i] != '\0')
  { j=0 ;
    k=i ;
    while (a[j] != '\0' && a[j] == b[k])
    { j++ ;
      k++ ;
    }
    if (a[j] == '\0') return 1 ;
    i++ ;
  }
  return 0 ;
}
```

2°)

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

#define MAXmot 50
#define MAXcar 20
```

```
typedef char Mot[MAXcar] ;

Mot dico[MAXmot] ;
int N ;

void lireDico(void)
{ N=0 ;
  fgets(dico[0], MAXcar, stdin) ;
  while (dico[N][0]!='.')
  { N++ ;
    if (N == MAXmot) return ;
    fgets(dico[N], MAXcar, stdin) ;
  }
}

void afficherDico(void)
{ int i;
  for (i=0 ; i<N ; i++)
    printf("%s\n", dico[i]) ;
}

void trierDico(void)
{ int i,j,k,jMin;
  Mot tmp;
  for (i=0 ; i<N-1 ; i++)
  { jMin = i;
    for (j=i+1 ; j<N ; j++)
      if (strcmp(dico[jMin], dico[j]) > 0) jMin=j;
    /* permutation : */
    strcpy(tmp,dico[jMin]);
    strcpy(dico[jMin],dico[i]);
    strcpy(dico[i], tmp);
  }
}

int main(void)
{ lireDico();
  trierDico();
  afficherDico();
  return 0;
}
```

3°) Accès via une table d'index

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

#define MAXmot 100
#define MAXcar 30
typedef char Mot[MAXcar] ;

Mot dico[MAXmot] ;
```

```
int N ;
int Index[MAXmot] ;

void lireDico(void)
{ N=0 ;
  fgets(dico[0], MAXcar, stdin) ;
  while (dico[N][0]!='.')
  { N++ ;
    if (N == MAXmot) return ;
    fgets(dico[N], MAXcar, stdin) ;
  }
}

void initIndex(void)
{ int i;
  for (i=0 ; i<MAXmot ; i++)
    Index[i] = i;
}

void afficherDico(void)
{ int i;
  for (i=0 ; i<N ; i++)
    printf("%s\n", dico[Index[i]]) ;
}

void trierDico(void)
{ int i,j,tmp,jMin;
  for (i=0 ; i<N-1 ; i++)
  { jMin = i;
    for (j=i+1 ; j<N ; j++)
      if (strcmp(dico[Index[jMin]], dico[Index[j]]) > 0) jMin=j;
    /* permutation des n° d'index : */
    tmp = Index[i];
    Index[i] = Index[jMin];
    Index[jMin] = tmp;
  }
}

int main(void)
{ lireDico();
  initIndex();
  trierDico();
  afficherDico();
  return 0;
}
```