```c
// $Id: cor.li012-03.c,v 1.1 2003/04/15 10:11:36 amsili Exp $

#include <strings.h>

#define LONG_MOT 22
// Longueur maximale d'un mot
#define NB_PREF 21
// Nombre maximal de préfixes pour un mot donné
#define MAX 6
// Nombre de mots traités
#define NB_PREF_TOTAL 140
// Majorant de MAX * NB_PREF

// ------------------------------------------------------------------------
// Question 1  : algo qui, à partir d'une chaîne de caractères s,
// affiche à l'écran tous les préfixes possibles de s (non triviaux)

void affprefixes(char s [])
{
  int i,j ;
  for (i=0 ; i<strlen(s)-1 ; i++) {
    for (j=0 ; j<=i ; j++) printf("%c",s[j]) ;
    printf("\n") ;
  }
}

// ------------------------------------------------------------------------
// Question 2 : structure de donnée plus algorithme pour stocker les
// préfixes possibles d'un mot s donné

typedef char t_liste_pref[NB_PREF][LONG_MOT] ;
t_liste_pref liste2 ;
// On définit le type "liste de préfixes" le la variable globale liste2

// La fonction renvoie le nombre de préfixes
int stocke_prefixes(char s [], t_liste_pref l)
{
  int i,j;
  for (i=0; i<strlen(s)-1 ; i++) {
    for (j=0 ; j<=i ; j++) l[i][j] = s[j] ;
    l[i][j] = '\0' ;
  }
  return i ;
}

// ------------------------------------------------------------------------
// Question 3 : supposons qu'on dispose d'un tableau tabs déjà
// rempli :
typedef char t_tableau_mots[MAX][LONG_MOT] ;
t_tableau_mots tabs ;
// Structure de données et algo pour conserver tous les préfixes de
// tous les mots de tabs.

// Le type "t_liste_pref" n'est pas suffisant : pour chaque mot, et
// donc pour chaque liste de préfixes, il faut connaître le nombre de
// préfixes. On définit donc un nouveau type "liste_pref", et une
// variable globale de ce type, 'tabs_pref'.
struct s_lst_pref {
  int          nb_pref ;
  t_liste_pref lst_pref ;
}  ;
typedef struct s_lst_pref liste_pref ;
liste_pref tabs_pref[MAX] ;
```



```c
void stocke_prefixes_tab(t_tableau_mots t, liste_pref tm[])
{
  int i ;
  for (i=0 ; i<MAX ; i++)
    tm[i].nb_pref = stocke_prefixes(t[i],tm[i].lst_pref) ;
}

// ------------------------------------------------------------------------
// Question 4 : il suffit de prendre un algo classique, par exemple le
// tri à bulles. Il faut disposer d'une fonction de comparaison des
// chaînes de caractères. Ici, on utilise la fonction prédéfinie
// strcmp. On a aussi besoin d'une fonction d'échange, 'swap'.
void swap(t_tableau_mots t, int i, int j)
{
  char temp[LONG_MOT] ;
  strcpy(temp, t[i]) ;
  strcpy(t[i], t[j]) ;
  strcpy(t[j], temp) ;
}

void tri(t_tableau_mots t)
{
  int changement=1, i, j ;

  for (i=0 ; i<MAX && changement ; i++) {
    changement = 0 ;
    for (j=MAX-1 ; j >= i+1 ; j--) {
      if (strcmp(t[j],t[j-1])<0) {
        swap(t, j, j-1) ;
        changement = 1 ;
      }
    }
  }
}

// ------------------------------------------------------------------------
// Question 5 : suppression des doublets.
// ------------------------------------------------------------------------
// N.B. Ceci ne peut fonctionner que sur un tableau *trié* : alors les
// doublets sont voisins. Puisqu'on supprime des éléments, la taille
// pertinente du tableau est modifiée, on renvoie la nouvelle valeur.
int supprime_doublets(t_tableau_mots t, int n)
{
  int i,j ;

  for (i=0 ; i<n-1 ; i++)
    while (strcmp(t[i], t[i+1])==0 && i<n-1) {
      for (j=i+1 ; j<n-1 ; j++)
        strcpy(t[j], t[j+1]) ;
      n-- ;
    }
  return n ;
}



// ------------------------------------------------------------------------
// Question 6 : fusion
// On suppose que l'on travaille sur un tableau de 'liste_pref', de
// taille MAX, et qu'on crée un tableau de mots de taille suffisante
// (NB_PREF_TOTAL)
typedef char t_tableau_final [NB_PREF_TOTAL][LONG_MOT] ;
t_tableau_final  total_pref ;
```

```c
// Fonctions intermédiaires pour faciliter la conception
int calcul_nb_total_pref(liste_pref tp[], int n)
{
  int i, nb ;
  for (i=0, nb=0 ; i<n ; i++) nb += tp[i].nb_pref ;
  return nb ;
}

// Pour suivre l'avancement de la fusion dans chaque case de
// tabs_pref, il faut un indice par case. Soit le tableau global
int indices[MAX] ;

void init_indices(int ti[], int n)
{
  int i ;
  for (i=0 ; i<n ; i++) ti[i] = 0 ;
}

int recherche_minimum(liste_pref t[], int n, int ind[])
{
  int i,j, i_champion ;
  char champion[LONG_MOT] ;
  for (i=0 ; ind[i] == t[i].nb_pref ; i++)
    ;
  i_champion = i ;
  strcpy(champion, t[i].lst_pref[ind[i]]) ;
  for (j=i+1 ; j<n ; j++) {
    if (ind[j] < t[j].nb_pref)
      if (strcmp(t[j].lst_pref[ind[j]], champion) < 0) {
        strcpy(champion, t[j].lst_pref[ind[j]]) ;
        i_champion = j ;
      }
  }
  return i_champion ;
}

int fusion(liste_pref t[], t_tableau_final res, int ind[])
{
  int i, j ;

  init_indices(ind,MAX) ;
  for (i=0 ; i<calcul_nb_total_pref(t, MAX) ; i++) {
    j = recherche_minimum(t,MAX,ind) ;
    strcpy(res[i], t[j].lst_pref[ind[j]]) ;
    ind[j]++ ;
  }
  return i ;
}
```



```c
// ------------------------------------------------------------------------
// Fonctions annexes ou de déboguage
// ------------------------------------------------------------------------
void aff_liste_prefixes(t_liste_pref l, int n)
{
  int i ;
  for (i=0 ; i<n ; i++)
    printf("pref no %2d : %s\n", i, l[i]) ;
}

void aff_tab_liste_pref(liste_pref t[], int n)
{
  int i ;
  for (i=0 ; i<n ; i++) {
    printf("----- Case n° %2d \n", i) ;
    aff_liste_prefixes(tabs_pref[i].lst_pref,
                       tabs_pref[i].nb_pref) ;
  }
}

void aff_tableau_mots(t_tableau_mots tab, int n)
{
  int i ;
  for (i=0 ; i<n ; i++)
    printf("Case n° %2d : '%s'\n", i, tab[i]) ;
}

void remplit_tableau_mots(t_tableau_mots t) // attention à MAX
{
  strcpy(t[0], "sage") ;
  strcpy(t[1], "douleur") ;
  strcpy(t[2], "tiens") ;
  strcpy(t[3], "toi") ;
  strcpy(t[4], "domino") ;
  strcpy(t[5], "t") ;
}

// ------------------------------------------------------------------------
// Programme principal   (Question 7)
// ------------------------------------------------------------------------
int main(int argc, char *argv[])
{
  int n, m, i ;

  if (argc >= 2) {
    affprefixes(argv[1]) ;                        // Question 1
    n = stocke_prefixes(argv[1],liste2) ;         // Question 2
    aff_liste_prefixes(liste2,n) ;              // Pour vérifier
    remplit_tableau_mots(tabs) ;                // Pour préparer la qst 3
    stocke_prefixes_tab(tabs, tabs_pref) ;        // Question 3
    printf("Les préfixes trouvés sont :\n") ;
    aff_tab_liste_pref(tabs_pref,MAX) ;         // Pour vérifier
    //    tri(tabs) ;                             // Question 4
    i = fusion(tabs_pref, total_pref, indices) ;  // Question 6
    m = supprime_doublets(total_pref, i) ;        // Question 5
    printf("Les préfixes triés et sans doublets sont : \n") ;
    aff_tableau_mots(total_pref, m) ;           // Pour vérifier
  }
}
```