TD 4 : Les tableaux

1. Parcours d'un tableau

On considère le tableau T de n entiers déjà initialisé. Ecrivez la suite d'instructions qui cherche la valeur et le rang du premier élément non nul dans les deux cas suivant :

- on est assuré que cet élément existe,
- il est possible que cet élément n'existe pas.

Que pensez-vous de l'idée suivante quand il est possible que cet élément n'existe pas :
On déclare T de taille n+1 et on affecte T[n] avec 1 (principe de la « sentinelle »)

2. Tableau partiellement rempli

Ecrire la suite d'instructions qui :

- déclare un tableau de 50 caractères
- lit une phrase au clavier terminée par un point et stocke cette phrase dans le tableau. Le point ne sera pas stocké dans le tableau et on prendra garde de stopper la lecture si le tableau est entièrement rempli. Une variable de type `int` mémorisera le nombre de caractères effectivement stockés.
- Affiche la phrase ainsi stockée (validation de la saisie)

3. Déplacer des éléments

On suppose le tableau précédent déjà initialisé.

Ecrire la suite d'instructions qui « tasse vers le bas » le tableau précédant en supprimant les blancs consécutifs pour ne conserver qu'un seul blanc entre les mots.

4. Matrice

Ecrire le programme qui déclare une matrice de caractères 4x7 , l'initialise avec des ' ' et des '*' de sorte à représenter la pyramide ci-dessous, et qui affiche son contenu.

```
   *
  ***
 *****
*******
```

Correction TD 4 : Les tableaux

1. Parcours d'un tableau

On considère le tableau T de n entiers déjà initialisés. Ecrivez la suite d'instructions qui cherche la valeur et le rang du premier élément non nul dans les deux cas suivant :

- on est assuré que cet élément existe,
- il est possible que cet élément n'existe pas.

```
#include <stdio.h>
#define MAX 10

void main(void)
{ int i;
  int T[MAX] = {0, 0, 0, 0, 0, 1, 2, 3, 4, 5}; /*exemple */
  i = 0;
  /* on est assuré que cet élément existe */
  while (T[i]==0) i++;
  printf("premier element non nul : T[%d] = %d\n", i,T[i] );
}

void main(void)
{ int i;
  int T[MAX] = {0, 0, 0, 0, 0, 1, 2, 3, 4, 5}; /*exemple */
  i = 0;
  /* il est possible que cet élément n'existe pas */
  while(i<MAX && T[i]==0) /* tester la borne avant la case! */
    i++;
  if (i<MAX)
    printf("premier element non nul : T[%d] = %d\n", i,T[i] );
  else
    printf("pas d'element non nul\n");
}

void main(void)
{ int i;
  int T[MAX+1]={0,0,0,0,0,1,2,3,4,5}; /*on crée une case de + */
  T[MAX] = 1 ; /* ajout de la sentinelle dans cette case */
  i = 0;
  while (T[i]==0) /*simplification du test grâce à la sentinelle*/
    i++;
  if (i<MAX)
    printf("premier element non nul : T[%d] = %d\n", i,T[i] );
  else /* on a "buté" sur la sentinelle */
    printf("pas d'element non nul\n");
}
```

2. Tableau partiellement rempli

Ecrire le programme qui :

- déclare un tableau de 50 caractères
- lit au clavier une phrase terminée par un point et stocke cette phrase dans le tableau
- on prendra garde de stopper la lecture si le tableau est entièrement rempli.

```
#define MAX 50
void main(void)
{ int n, i;  // nbre de caractères stockés et indice courant
  char carcou ;     // caractère courant
  int T[MAX] ;
  n = 0;
  carcou = getchar() ;  /* ou scanf("%c", &carcou ) ; */
  while ((n<MAX) && (carcou !='.'))
  { T[n++] = carcou ; // on stocke carcou puis on incrémente n
    carcou = getchar() ;
  }
  /* réécriture pour contrôler la saisie */
  for (i=0 ; i<n ; i++)
    printf("%c", T[i]) ;
  printf(".\n") ;
}
```

3. Déplacer des éléments

On suppose le tableau précédent déjà initialisé.

Ecrire la suite d'instructions qui « tasse vers le bas » le tableau précédant en supprimant les blancs consécutifs pour ne conserver qu'un seul blanc entre les mots.

```
#include <stdio.h>
#define MAX 50

void main(void)
{ int i,nbCar;      // indice courant et nbre de car. Stockés
  int j ;           // indice de recopie
  char P[MAX], car, carPred;
  nbCar = 0;
  printf("donnez votre phrase\n");
  scanf("%c", &car);
  while (nbCar<MAX && car != '.')
  { P[nbCar] = car;
    nbCar++;
    scanf("%c", &car);
  }
  /* etape 2 : supprimer les blancs consécutifs */
  j=0;
```

```
  carPred = ' '; /* joue le role du predecesseur */
  for (i=0 ; i<nbCar ; i++)
  { if (P[i] != ' ' || carPred!= ' ')
    { P[j] = P[i]; // recopie en j
      j++;
    }
    carPred = P[i];
  }
  nbCar = j;
  for (i=0 ; i<nbCar ; i++)
    printf("%c", P[i]);
  printf("\n");
}
```

4. Matrice

Ecrire le programme qui déclare une matrice de caractères 4x7 , l'initialise avec des ' ' et des '*' de sorte à représenter la pyramide ci-dessous, et qui affiche son contenu.

```
   *
  ***
 *****
*******
/* Solution avec allocation dynamique de mémoire */
#include <stdio.h>
#include <stdlib.h>

void main(void)
{ int i,j, hauteur, largeur;
  int debEtoile, finEtoile ;
  typedef char *ligne;
  ligne *pyramide ; /* ou char **pyramide ; */
  printf("donnez la hauteur de la pyramide\n");
  scanf("%d", &hauteur);
  /* allocation de la matrice */
  pyramide = malloc(sizeof(ligne) * hauteur) ;
  largeur = 2*hauteur-1;
  for (i=0 ; i<hauteur ; i++)
    pyramide[i] = malloc(sizeof(char) * largeur) ;

  debEtoile = 0 ;
  finEtoile = largeur ;
  for (i=0 ; i<hauteur ; i++)
  { j=0 ;
    while (j<debEtoile)
      pyramide[i][j++] = ' ';
    while (j<finEtoile)
      pyramide[i][j++] = '*';
```

```
    while (j<largeur)
      pyramide[i][j++] = ' ';
    // début et fin des * dans la ligne suivante
    debEtoile++ ;
    finEtoile-- ;
  }

  /* la base de la pyramide est à la ligne 0 */
  /* cette base doit etre affichee en dernier */
  i=hauteur ;
  while (i--) /*1ere itération avec i=hauteur-1, arrêt pour i==0*/
  { for(j=0 ; j<largeur ; j++)
      printf("%c",pyramide[i][j]);
    printf("\n");
  }
}
```