# Correction TD-TME1

## TD

## Exercice 1

**1)**

```
void min_max_moy(int tab[], int taille, int *min, int *max, float *moy)
```

**2)**

```
#include<stdio.h>
#define TAILLE 6
void min_max_moy(int tab[], int taille, int *min, int *max, float *moy)
	{
		unsigned int i;
		int somme=tab[0];
		*min=tab[0];
		*max=tab[0];
		for (i=1;i<taille;i++) {
		if (tab[i]<*min) {
			*min=tab[i];
		}
		if (tab[i]>*max) {
			*max=tab[i];
		}
		somme+=tab[i];
	}
	*moy=(float)somme/(float)taille;
}
int main() {
	int t[TAILLE]={1,2,3,4,0,-1};
	int min,max;
	float moy;
	unsigned int i;
	min_max_moy(t,TAILLE,&min,&max,&moy);
	printf("Tableau: [ %d",t[0]);
	for (i=1;i<TAILLE;i++) {
		printf(",%d",t[i]);
	}
	printf("]\n");
	printf("Max: %d Min: %d Moy: %f\n",max,min,moy);
}
```

**3)**

```
void min_max_som(int tab[], int taille, int *min, int *max, int *som) {
	if (taille>1) {
	min_max_som(tab+1,taille-1,min,max,som);
	*som+=tab[0];
	*min=*min>tab[0]?tab[0]:*min;
	*max=*max<tab[0]?tab[0]:*max;
	/* nous avons utilise ici l'ecriture condensee. Pour rappel:
```

```
        a=b>c?d:e;
        est equivalent a:
        if (b>c)
                a=d
        else
                a=e
        */
        }
        else {
                *min=tab[0];
                *max=tab[0];
                *som=tab[0];
        }
}
void min_max_moy(int tab[], int taille, int *min, int *max, float *moy) {
        int som;
        min_max_som(tab,taille,min,max,&som);
        *moy=(float)som/(float)taille;
}
```

**4)**

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
int main() {
        int *t;
        int min,max;
        float moy;
        unsigned int i,j;
        for (i=1;i<=100;i++) {
                t=(int *)malloc(i*sizeof(int));
                if (t==NULL) {
                        printf("Probleme lors de l'allocation du tableau\n");
                        exit(1);
                }
                for (j=0;j<i;j++) {
                        t[j]=j+1;

                        /* j+1 pour que le tableau contienne des valeurs
                        entre 1 et n compris et non entre 0 et n-1 */
                }
                min_max_moy(t,i,&min,&max,&moy);
                if (min!=1) {
                        printf("Erreur. Calcul sur un tableau de taille %d ",i);
                        printf("le minimum est de %d au lieu de %d\n",min,1);
                }
                if (max!=i) {
                        printf("Erreur. Calcul sur un tableau de taille %d ",i);
                        printf("le maximum est de %d au lieu de %d\n",max,1);
                }
```

```
                if (fabs(moy-(float)(i+1)/2.0)>0.0001) {
                        /* la somme des n premiers entiers vaut n*(n+1)/2, donc
                        la moyenne vaut (n+1)/2.
                        Comme il s'agit de nombres reels et pour eviter tout
                        probleme d'arrondi, il vaut mieux tester si la difference
                        (en valeur absolue) est inferieure a un seuil arbitrairement
                        petit.
                        */
                        printf("Erreur. Calcul sur un tableau de taille %d ",i);
                        printf("la moyenne est de %f au lieu de %f\n",moy,(float)(i+1)/
(float)2.0);
                }
                free(t);
        }
}
```

## Exercice 2

**1)**

```
pc = malloc((nb_char + 1) * sizeof(char));
```

**2)**

```
int compte_mots_chaine(char *chaine) {
        int nb_mots=0;
        while (*chaine)
        {
                if(*chaine == ' ')
                {
                        chaine++;
                        continue;
                }
                nb_mots++;
                while((*chaine != ' ') && (*chaine)) chaine++;
        }
        return nb_mots;
}
```

**3)**

```
int compte_mots(char **ptab_mots);
```

L'argument est un pointeur sur un tableau contenant des pointeurs sur des char, donc un pointeur sur un pointeur sur un char...

**4)**

```
int compte_mots(char **ptab_mots)
{
        int cpt=0;
        while(ptab_mots[cpt]) cpt++;
        return cpt;
}
```

**5)**

```
void detruit_tab_mots(char **ptab_mots)
{
	int i=0;
	if (ptab_mots)
		while(ptab_mots[i])
			free(ptab_mots[i++]);
	free(ptab_mots);
}
```

**6)**

```
char **ptab;
...
ptab = (char **)malloc((n + 1) * sizeof(char *));
```

**7)**

```
char **decompose_chaine(char *chaine)
{
	char *pc= chaine;
	int nb_mots=0;
	char **ptab;
	char *psrc_mot;
	int ind_mot=0;
	//comptages des mots
	nb_mots=compte_mots_chaine(chaine);
	if (nb_mots == 0)
		return NULL;
	// allocation du tableau
	ptab = malloc((nb_mots + 1) * sizeof(char *));
	ptab[nb_mots] = NULL;
	// copie des mots
	pc=chaine;
	while (*pc)
	{
		if(*pc == ' ')
		{
			pc++;
			continue;
		}
		psrc_mot = pc;
		while((*pc != ' ') && (*pc)) pc++;
		//allocation du mot
		ptab[ind_mot] = malloc((pc - psrc_mot + 1)* sizeof(char));
		//copie du mot
		memcpy(ptab[ind_mot], psrc_mot, pc - psrc_mot);
		//insertion du marqueur de fin de chaine
		*(ptab[ind_mot] + (pc - psrc_mot)) = '\0';
		ind_mot++;
	}
	return ptab;
}
```

## TME
## Exercice 3

**1)**

```
int main(){
	char chaine[]="mot1 et mot2 et mot3";
	printf("Chaine avant decomposition :\n%s\n",chaine);
	printf("Nombre de mots : %d\n", compte_mots_chaine(chaine));
	return 0;
}
```

**2)**

Un tableau statique `a 2 dimensions n'est pas un double pointeur : char s ptab mots[6][5] n'est pas du
même type que char **ptab mots.
Pour respecter les prototypes, il faut donc d'abord d´eclarer un tableau de pointeurs et affecter des adresses valides aux différentes cases du tableau.

```
int main(){
	char *s_ptab_mots[6]={"mot1", "et", "mot2", "et", "mot3", NULL};
	printf("nombre de mots dans s_ptab_mot : %d\n",
compte_mots(s_ptab_mots));
	return 0;
}
```

**3)**

```
void affiche_tab_mots(char **ptab_mots)
{
	int i=0;
	while(ptab_mots[i])
	printf("%s\n", ptab_mots[i++]);
}
```

La fonction se contente de parcourir le tableau jusqu'à atteindre la case contenant la valeur NULL. A chaque fois, la valeur ptab[i] est affichée sur une ligne. Attention au i++ ! Le printf peut être remplacé de façon équivalente par les deux instructions suivantes :

```
printf("%s\n", ptab_mots[i]);
i=i+1;
```

**4)**

```
char *compose_chaine(char **ptab_mots)
{
	int i=0, taille=1;
	char *pc, *tmp_pc, *pmot;
	if (ptab_mots == NULL)
		return NULL;
	//calcul de la longueur de la chaine resultat
	while(ptab_mots[i])
		taille += strlen(ptab_mots[i++]) +1; //+1 pour les espaces
```

```
	pc = malloc(taille * sizeof(char));
	//copie des mots
	tmp_pc = pc;
	i=0;
	while(ptab_mots[i])
	{
		pmot = ptab_mots[i];
		while(*pmot)
			*tmp_pc++ = *pmot++;
		*tmp_pc++ = ' ';
		i++;
	}
	*tmp_pc = '\0';
	return pc;
}
```

**5)**

Il suffit d'ajouter les instructions suivantes :

```
	chaine_composee = compose_chaine(ptab_mots);
	if (chaine_composee)
		printf("Apres recomposition :\n%s\n", chaine_composee);
```

et, à la fin du programme :

```
	free(chaine_composee);
```

## Exercice 4

**1)**

```
char **reduit_tab_mots(char **ptab_mots)
{
	int i=0, j;
	if (ptab_mots)
		while(ptab_mots[i])
		{
			j = i+1;
			while (ptab_mots[j])
			{
				if (ptab_mots[i] == ptab_mots[j])
				{
					j++; continue;
				}
				if (strcmp(ptab_mots[i], ptab_mots[j]))
				{
					j++; continue;
				}
				free(ptab_mots[j]);
				ptab_mots[j] = ptab_mots[i];
			}
			i++;
		}
	return ptab_mots;
```

```
}
```

**2)**
Oui car il ferait plusieurs fois appel à la fonction free pour les mêmes adresses, le comportement serait alors indéterminé cf man free. Pour éviter le problème, une solution pourrait consister à, pour chaque case du tableau, parcourir l'intégralité du tableau pour rechercher et marquer d'éventuelles autres occurrences en les mettant à NULL, c'est une opération assez lente nécessitant n(n+1)/2 comparaisons (si n est la taille du tableau).