# TD/TP02 Solution

## Solution 01

```
#include <iostream>
using namespace std;
// Fonction pour échanger le contenu de deux entiers en utilisant des pointeurs
void swap(int *x, int *y) {
	int temp = *x;
	*x = *y;
	*y = temp;
}
int main() {
	int a, b;
	// Saisie des valeurs de a et b
	cout << "Entrez la valeur de a : ";  cin >> a;
	cout << "Entrez la valeur de b : ";  cin >> b;
	// Affichage des valeurs avant l'échange
	cout << "\nAvant l'echange :" << endl;
	cout << "a = " << a << ", b = " << b << endl;
	// Appel de la fonction swap
	swap(&a, &b);
	// Affichage des valeurs après l'échange
	cout << "\nApres l'echange :" << endl;
	cout << "a = " << a << ", b = " << b << endl;
	return 0;
}
```

## Solution 02

```
#include <iostream>
int countA(const char* str) {
	int count = 0;
	const char* ptr = str;
	while (*ptr != '\0') {  // Parcourt jusqu'à la fin de la chaîne
		if (*ptr == 'A') count++;
		ptr++;  // Avance le pointeur
	}
	return count;
}
int main() {
	const char* str = "Abracadabra Amazing Apple";
	int occurrences = countA(str);
	std::cout << "Nombre d'occurrences de 'A': " << occurrences << std::endl;
	return 0;
}
```

## Solution 03

On peut commencer par remplir t1 de zéros, avant d'y recopier les éléments positifs de t2 :

```
	int i, j ;
	for (i=0 ; i<10 ; i++) t1[i] = 0 ;
		/* i sert à pointer dans t1 et j dans t2 */
```

# TD/TP02 Solution

```
for (i=0, j=0 ; j<10 ; j++)
     if (t2[j] > 0) t1[i++] = t2[j] ;
```

Mais on peut recopier d'abord dans t1 les éléments positifs de t2, avant de compléter éventuellement par des zéros. Cette seconde formulation, moins simple que la précédente, se révélerait toutefois plus efficace sur de grands tableaux :

```
int i, j ;
for (i=0, j=0 ; j<10 ; j++)
     if (t2[j] > 0) t1[i++] = t2[j] ;
for (j=i ; j<10 ; j++) t1[j] = 0 ;
```

## Solution 04

Étant donné que la fonction doit être en mesure de modifier la valeur de son second argument, il est nécessaire que ce dernier soit transmis par référence.

```
#include <iostream>
using namespace std ;
void ajoute (int exp, int & var){
      var += exp ;
      return ;
}
main(){
      void ajoute (int, int &) ;
int n = 12 ;
int p = 3 ;
cout << "Avant, n = " << n << "\n" ;
ajoute (2*p+1, n) ;
cout << "Après, n = " << n << "\n" ;
}
```

## Solution 05

En C++, par défaut, les arguments sont transmis par valeur. Mais, dans le cas d'un tableau, cette valeur, de type pointeur, n'est rien d'autre que son adresse. Quant à la transmission par référence, elle n'a pas de signification dans ce cas. Nous n'avons donc aucun choix en ce qui concerne le mode de transmission de notre tableau.

En ce qui concerne le nombre d'éléments (de type int), nous le transmettrons classiquement par valeur.

L'en-tête de notre fonction pourra se présenter sous l'une des formes suivantes :

```
float somme (float t[], int n)
float somme (float * t, int n)
float somme (float t[5], int n)     // déconseillé car laisse croire que t
                                    // est de dimension fixe 5
```

En effet, la dimension réelle de **t** n'a aucune incidence sur les instructions de la fonction elle-même (elle n'intervient pas dans le calcul de l'adresse d'un élément du tableau1 et elle ne sert pas à « allouer » un emplacement puisque le tableau en question aura été alloué dans la fonction appelant somme).

Voici ce que pourrait être la fonction demandée :

```
float somme (float t[], int n) {    // on peut écrire somme (float * t,...
                                    // ou encore somme (float t[4], ...
                                    // mais pas somme (float t[n], ...
      int i ;
      float s = 0 ;
```

# TD/TP02 Solution

```
        for (i=0 ; i<n ; i++)
              s += t[i] ; // on pourrait écrire s += * (t+i) ;
        return s ;
  }
```

Pour ce qui est du programme d'utilisation de la fonction somme, on peut, là encore, écrire le « prototype » sous différentes formes :

```
  float somme (float [], int ) ;
  float somme (float * , int ) ;
  float somme (float [5], int ) ;     // déconseillé car laisse croire que t
                                      // est de dimension fixe 5
```

Voici un exemple d'un tel programme :

```
  #include <iostream>
  using namespace std ;
  main(){
        float somme (float *, int) ;
        float t[4] = {3, 2.5, 5.1, 3.5} ;
        cout << "somme de t : " << somme (t, 4) ;
  }
```

## Solution 06

La meilleure solution consiste à prévoir, au sein de la fonction en question, une variable de classe statique. Elle sera initialisée une seule fois à zéro (ou à toute autre valeur éventuellement explicitée) au début de l'exécution du programme. Ici, nous avons, de plus, prévu un petit programme d'essai.

```
#include <iostream>
using namespace std ;
void fcompte (void){
      static int i ; // il est inutile, mais pas interdit, d'écrire i=0
      i++ ;
      cout << "appel numéro " << i << "\n" ;
}
/* petit programme d'essai de fcompte */
main(){
      void fcompte (void) ;
      int i ;
      for (i=0 ; i<3 ; i++) fcompte () ;
}
```

Là encore, la démarche consistant à utiliser comme compteur d'appels une variable globale (qui devrait alors être connue du programme utilisateur) est à proscrire.

## Solution 07

```
#include <iostream>
#include <string>
using namespace std;
int main() {
      // Première partie : Saisie de 10 caractères
      char texte[10];
      int count_e = 0;
      cout << "Saisir un texte de 10 caracteres : ";
      for (int i = 0; i < 10; i++) {
```

# TD/TP02 Solution

```
            cin >> texte[i];
            if (texte[i] == 'e') {
                  count_e++;
            }
      }
      cout << "Le nombre de lettres 'e' dans le texte est : " << count_e << endl;
      // Deuxième partie : Saisie d'un texte avec taille spécifiée par l'utilisateur
      int taille;
      cout << "\nCombien de caracteres souhaitez-vous traiter ? "; cin >> taille;
      // Allocation dynamique de mémoire pour le texte
      char *texte_dynamique = new char[taille];
      int count_e_dynamique = 0;
      cout << "Saisir un texte de " << taille << " caracteres : ";
      for (int i = 0; i < taille; i++) {
            cin >> texte_dynamique[i];
            if (texte_dynamique[i] == 'e') count_e_dynamique++;
      }
      cout << "Le nombre de lettres 'e' dans le texte dynamique est : " <<
      count_e_dynamique << endl;

      // Libération de la mémoire allouée dynamiquement
      delete[] texte_dynamique;

      return 0;
}
```

## Solution 08

```
#include <iostream>
using namespace std;

int main() {
      // Déclaration d'un entier
      int a;
      cout << "Donner un entier : ";  cin >> a;
      // Déclaration d'une référence vers cet entier
      int &ref_a = a;
      // Déclaration d'un pointeur vers cet entier
      int *p_a = &a;
      // Affichage des valeurs des variables
      cout << "\nLe contenu des variables:" << endl;
      cout << "- variable a : " << a << endl;
      cout << "- variable ref_a : " << ref_a << endl;
      cout << "- variable p_a : " << p_a << endl;
      // Affichage des adresses des variables
      cout << "\nLeur adresse:" << endl;
      cout << "- adresse de a : " << &a << endl;
      cout << "- adresse de ref_a : " << &ref_a << endl;
      cout << "- adresse de p_a : " << &p_a << endl;
      // Affichage de la valeur pointée par p_a
      cout << "\nLa valeur pointee : " << *p_a << endl;
      return 0;
```

# TD/TP02 Solution

```
}
```

## Solution 09

```
#include <iostream>
using namespace std;
// Fonction pour incrémenter en passant l'adresse
void incrementer(int *a) { (*a)++; }
// Fonction pour permuter les valeurs en passant l'adresse
void permuter(int *a, int *b) { int temp = *a;  *a = *b; *b = temp; }
// Fonction pour incrémenter en utilisant le passage par référence
void incrementerRef(int &a) { a++; }
// Fonction pour permuter les valeurs en utilisant le passage par référence
void permuterRef(int &a, int &b) { int temp = a;  a = b;  b = temp; }
int main() {
      int A, B;
      // Saisie des valeurs de A et B
      cout << "Donner la valeur de A: ";  cin >> A;
      cout << "Donner la valeur de B: ";  cin >> B;
      // Utilisation du passage par adresse
      cout << "\n- Appel par Adresse" << endl;
      incrementer(&A);
      cout << "la valeur de A apres incrementation : " << A << endl;
      permuter(&A, &B);
      cout << "la valeur de A apres permutation : " << A << endl;
      cout << "la valeur de B apres permutation : " << B << endl;
      // Utilisation du passage par référence
      cout << "\n- Appel par Référence" << endl;
      cout << "la valeur de A est : " << A << endl;
      cout << "la valeur de B est : " << B << endl;
      incrementerRef(A);
      cout << "la valeur de A apres incrementation : " << A << endl;
      permuterRef(A, B);
      cout << "la valeur de A apres permutation : " << A << endl;
      cout << "la valeur de B apres permutation : " << B << endl;
      return 0;
}
```