Exercice1

Donnez le résultat d'exécution des fonctions suivantes. Représentez schématiquement l'état de la mémoire.

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

void exemple1() {
      int * p; int x=17;
      p=&x; x++; (*p)++;
      cout<<"x="<<x<<"   *p="<<*p<<endl;
}
void exemple2() {
      char *pc, *pb, car;
      car='t'; pc=&car; pb=pc; car=car-'a'+'A';
      cout<<car<<*pc<<*pb<<endl;
}
void exemple3() {
      char mot[10];
      strcpy(mot,"papillon"); cout<<"*mot="<<*mot<<endl;
      *mot=*(mot+2)='t'; cout<<mot<<endl;
}
```

Exercice2

Donnez le résultat d'exécution du programme suivant:

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

void main() {
       const MAX=50;
       char nom1[MAX],nom2[MAX]; char * nom3;
       cout<<"saisir un nom: "; cin.get(nom1,MAX);
       nom3=nom1; strcpy(nom2,nom1);
       cout<<"nom1="<<nom1<<"\nnom2="<<nom2<<"\nnom3="<<nom3<<endl;
       cout<<"saisir un autre nom: "; cin.get(nom1);
       cout<<"nom1="<<nom1<<"\nnom2="<<nom2<<"\nnom3="<<nom3<<endl;
}
```

Exercice3

1-Ecrire un programme permettant de saisir le contenu d'un tableau de nombres réels. Le programme affichera ensuite les valeurs entrées. Le nombre de valeurs à entrer (nombre d'éléments) est demandé à l'utilisateur

2- Créer la procédure **AfficherTableau** qui permet d'afficher tous les éléments d'un tableau (le nombre d'éléments du tableau doit nécessairement être passé en paramètre). Modifier le programme afin d'utiliser cette fonction.

3- Créer la fonction **SaisirNbElements** qui demande à l'utilisateur le nombre d'éléments à entrer. Utiliser cette fonction dans le programme précédent.

4- Ajouter au programme précédent la procédure **SaisirTableau**.

5- Modifier le programme afin d'utiliser une allocation dynamique pour le tableau : le nombre d'éléments à entrer sera alors égal à la taille du tableau.

Exemple d'allocation dynamique :

```
int nb ; double *tab = NULL ;// déclaration + affectation du pointeur
cout << "Nombre d'éléments ? ";
cin >> nb ;
```

tab = new double[nb] ; // allocation de l'espace mémoire
… // utilisation du tableau
…
delete[] tab ; // libération de l'espace mémoire

6 Faire la fonction **AdditionneTableaux**, permettant de faire la somme terme à terme de deux tableau de même taille. Tester cette fonction avec 2 tableaux (tab1 + tab2) puis avec 3 tableaux (le but est de faire la somme termes à termes de trois tableaux).

Exercice 4 : Plus petit diviseur premier

Écrire une fonction qui retourne le plus petit diviseur premier d'un nombre.

Exercice 5 : Fusion de deux listes

Ecrire une fonction **fusion** permettant de fusionner deux listes triées passées en paramètre.

Par exemple :

```
T1=[0, 2, 4, 6]
T2=[1, 3, 5]
>>>fusion(T1,T2)
[0, 1, 2, 3, 4, 5, 6]
```

Exercice 6 : Palindrome

Ecrire une fonction permettant de déterminer si une chaîne de caractères est ou non un palindrome (i.e. pouvant être lue indifféremment de la gauche vers la droite ou de la droite vers la gauche).

Exercice 7 : Somme de Fibonacci

1) Implémenter la fonction de Fibonacci de la manière suivante :

```
def fibonacciRecursive(n):
  if n==0 or n==1:
    return 1
  else:
    return fibonacciRecursive(n-1) + fibonacciRecursive(n-2)
```

Exercice 8 : Tri Rapide

Principe du tri rapide :

- ❑ On choisit le pivot (le dernier élément du tableau)
- ❑ Le partitionnement peut être fait en temps linéaire, en place. La mise en œuvre la plus simple consiste à parcourir le tableau du premier au dernier élément, en formant la partition au fur et à mesure : à gauche on a l'ensemble d'élément inférieur au pivot et à droite on a l'ensemble d'élément supérieur ou égale au pivot.

- Pour chaque sous ensemble, on définit un nouveau pivot et on répète l'opération de partitionnement. Ce processus est répété récursivement, jusqu'à ce que l'ensemble des éléments soit trié.
- Ecrire la faction du tri rapide.
- Exemple
-