Atelier 2: Notions avancées

Exercice 1 :

Écrire une fonction, sans argument ni valeur de retour, qui se contente d'afficher, à chaque appel, le nombre total de fois où elle a été appelée sous la forme : ***appel numéro 3***

Réponse :

```
#include <iostream>
using namespace std;
int a=0;
void appel(){
    a+=1;
    cout<<"Appel numero "<<a<<endl;
}
int main()
{
    appel();
    appel();
    appel();
    return 0;
}
```

Exercice 2 :

Écrire 2 fonctions à un argument **entier** et une valeur de retour **entière** permettant de préciser si l'argument reçu est multiple de **2** (pour la première fonction) ou multiple de **3** (pour la seconde fonction).

Utiliser ces deux fonctions dans un petit programme qui lit un nombre entier et qui précise s'il est pair, multiple de 3 et/ou multiple de 6, comme dans cet exemple (il y a deux exécutions) :

```
donnez un entier : 9
il est multiple de 3
---------------
donnez un entier : 12
il est pair
il est multiple de 3
il est divisible par 6
```

Réponse :

```
#include <iostream>
using namespace std;
int nbr_pair(int m){
    if(m%2==0){
        cout<<"il est pair"<<endl;
        return m;
    }else
        return m;
}
```

```
int nbr_mult3(int m){
    if(m%3==0){
        cout<<"il est multiple de 3"<<endl;
        return m;
    }else
        return m;
}
int main()
{
    int n,a;
    cout << "Donner un nombre : ";
    cin>>n;
    a=nbr_pair(n);
    a=nbr_mult3(n);
    if(a%2==0 && a%3==0)
        cout<<"il est divisible par 6"<<endl;

    return 0;
}
```

Exercice 3 :

Écrire, de deux façons différentes, un programme qui lit **10 nombres entiers** dans un tableau avant d'en rechercher le plus grand et le plus petit :

a. en utilisant uniquement le « *formalisme tableau* » ;
b. en utilisant le « *formalisme pointeur* », à chaque fois que cela est possible.

Réponse :

```
#include <iostream>
using namespace std;
int main()
{
    int i,max,min;
    int T[10];
    int *p=T;
    //En Utilisant le formalisme tableau//
    for(i=0;i<10;i++){
        cout<<"Donner le nombre de la case "<<i+1<<" du tableau : ";
        cin>>T[i];
    }
    max=T[0];
    for(i=1;i<10;i++){
        if(max<T[i])
            max=T[i];
    }
    min=T[0];
    for(i=1;i<10;i++){
        if(min>T[i])
            min=T[i];
    }
```

```
    cout<<"\nLe plus grand nombre de ce tableau est : "<<max<<endl;
    cout<<"Le plus petit nombre de ce tableau est : "<<min<<endl;
    //En Utilisant le formalisme pointeur//
    for(p=T;p<T+10;p++){
        cout<<"Donner le nombre de la case "<<p-T+1<<" du tableau : ";
        cin>>*p;
    }
    p=T;
    max=*p;
    for(p=T;p<T+10;p++){
        if(max<*p)
            max=*p;
    }
    min=*p;
    for(p=T;p<T+10;p++){
        if(min>*p)
            min=*p;
    }
    cout<<"\nLe plus grand nombre de ce tableau est : "<<max<<endl;
    cout<<"Le plus petit nombre de ce tableau est : "<<min<<endl;
    return 0;
}
```

Exercice 4 :

Écrire un programme **allouant dynamiquement** un emplacement pour un **tableau d'entiers**, dont la taille est fournie en donnée.

1. Utiliser ce tableau pour y placer des nombres entiers lus également en donnée.
2. Créer ensuite dynamiquement un nouveau tableau destiné à recevoir les carrés des nombres contenus dans le premier.
3. Supprimer le premier tableau, afficher les valeurs du second et supprimer le tout.

Réponse :

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    int Taille,i;
    cout<<"Donner la taille du tableau : ";
    cin>>Taille;
    int *T1=new int [Taille];
    int *T2=new int [Taille];
    for(i=0;i<Taille;i++){
        cout<<"Donner la valeur de la case "<<i+1<<" du premier tableau : ";
        cin>>T1[i];
    }
    for(i=0;i<Taille;i++){
        T2[i]=pow(T1[i],2);
    }
```

```
    delete[] T1;
    cout<<"\nLes valeurs du deuxieme tableuax sont :"<<endl;
    for(i=0;i<Taille;i++){
        cout<<"case "<<i+1<<" : "<<T2[i]<<endl;
    }
    delete[] T2;
    return 0;
}
```

Exercice 5 :

Ecrire un programme C++ qui :

1. déclare un entier a;
2. déclare une référence vers cet entier ref_a;
3. déclare un pointeur vers cet entier p_a;
4. affiche les variables, leurs adresses, la valeur pointée.

Réponse :

```
#include <iostream>
using namespace std;
int main()
{
    int a=2;
    int &ref_a=a;
    int *p_a=&a;
    cout<<" Les variables sont : a, ref_a, p_a "<<endl;
    cout<<" L'adresse de a est : "<<&a<<endl;
    cout<<" L'adresse de ref_a est : "<<&ref_a<<endl;
    cout<<" L'adresse de pointeur est : "<<&p_a<<endl;
    cout<<" La valeur pointee par p_a est : "<<*p_a<<endl;
    return 0;
}
```

Exercice 6 :

Écrire une fonction nommée **incrementer**() permettant d'incrémenter la valeur d'une variable passée en paramètre et une fonction nommée **permuter**() permettant d'échanger les contenus de 2 variables de type int fournies en argument :

1. en transmettant l'adresse des variables concernées (seule méthode utilisable en C) ;
2. en utilisant la transmission par référence.

Dans les deux cas, écrire un programme (**main**) qui teste les deux fonctions.

Réponse :

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

void incrementer_adresse(int *N){
    (*N)++;
}
```

```
void permuter_adresse(int *N, int *M){
    int temp;
    temp=*N;
    *N=*M;
    *M=temp;
}
void incrementer_reference(int &N){
    N++;
}
void permuter_reference(int &N, int &M){
    int temp;
    temp=N;
    N=M;
    M=temp;
}
int main()
{
    int N=1,M=5,A=1,B=5;
    //En transmettant l'adresse des variables concernées//
    incrementer_adresse(&N);
    cout<<"La valeur de N apres incrementation par adresse est : "<<N<<endl;
    permuter_adresse(&N,&M);
    cout<<"Apres permutation par adresse, la valeur de N est "<<N<<" et la valeur
de M est "<<M<<endl;
    //En utilisant la transmission par référence//
    incrementer_reference(A);
    cout<<"\nLa valeur de A apres incrementation par reference est : "<<A<<endl;
    permuter_reference(A,B);
    cout<<"Apres permutation par reference, la valeur de A est "<<A<<" et la
valeur de B est "<<B<<endl;
    return 0;
}
```

Exercice 7 :

Ecrire un programme qui demande à l'utilisateur de taper 10 entiers qui seront stockés dans un tableau. Le programme doit trier le tableau par ordre croissant et doit afficher le tableau.
Algorithme suggéré (tri bulle) :
On parcourt le tableau en comparant t[0] et t[1] et en échangeant ces éléments s'ils ne sont pas dans le bon ordre.

1. On recommence le processus en comparant t[1] et t[2],... et ainsi de suite jusqu'à t[8] et t[9].
2. On compte lors de ce parcours le nombre d'échanges effectués.
3. On fait autant de parcours que nécessaire jusqu'à ce que le nombre d'échanges soit nul : le tableau sera alors trié.

Réponse :

```
#include <iostream>
using namespace std;
int main()
{
    int i,temp,echange;
    int T[10];
    for(i=0;i<10;i++){
```

```
        cout<<"Donner le nombre de la case "<<i+1<<" du tableau : ";
        cin>>T[i];
    }
    cout<<"Le tableau est :"<<endl;
    for(i=0;i<10;i++){
        cout<<T[i]<<" |";
    }
    do{
    echange=0;
    for(i=0;i<9;i++){
        if(T[i]>T[i+1]){
            temp=T[i];
            T[i]=T[i+1];
            T[i+1]=temp;
            echange=1;
            }
      }
    }while(echange);
    cout<<"\nApres le tri en ordre croissant, le tableau devient :"<<endl;
    for(i=0;i<10;i++){
        cout<<T[i]<<" |";
    }
    return 0;
}
```