CHAPITRE 7

LES FONCTIONS

En langage C++ les sous-programmes s'appellent des **fonctions.**

Une fonction possède un et un seul point d'entrée, mais éventuellement plusieurs points de sortie (à l'aide du mot **return**).

Le programme principal (**void main()**), peut être considéré comme une fonction parmi les autres. C'est le point d'entrée du programme.

Une variable connue uniquement d'une fonction ou est une variable locale.
Une variable connue de tout le programme est une variable globale.

FONCTIONS SANS PASSAGE D'ARGUMENTS ET NE RENVOYANT RIEN AU PROGRAMME.

Ce type de fonction est appelé depuis le programme principal ou un autre sous-programme. Elle est alors exécutée, mais le programme appelant n'en retire aucune valeur à exploiter.

Exemple à expérimenter:

Exercice VII_1:

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

void bonjour()       // declaration de la fonction
{
cout<<"bonjour\n";
}

void main()             // programme principal
{
bonjour();   // appel de la fonction, et donc exécution a partir de sa 1ere ligne
cout<<"POUR CONTINUER FRAPPER UNE TOUCHE: ";
getch();
}
```

Conclusion :
Il ne faut pas confondre **déclaration** avec **exécution**.
Les fonctions sont déclarées au début du fichier source. Mais elles ne sont exécutées que si elles sont appelées par le programme principal ou le sous-programme.
Une fonction peut donc être décrite en début de programme mais ne jamais être excutée.

*L'expression **void bonjour()** est appelé le prototype de la fonction " bonjour "*

Exemple à expérimenter:

Exercice VII_2:

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

void bonjour()       // declaration de la fonction
{
cout<<"bonjour\n";
}

void coucou()        // declaration de la fonction
{
bonjour();     // appel d'une fonction dans une fonction
cout<<"coucou\n";
}

void main()             // programme principal
{
coucou();   // appel de la fonction
cout<<"POUR CONTINUER FRAPPER UNE TOUCHE: ";
getch();
}
```

Conclusion :

Un programme peut contenir autant de fonctions que nécessaire.
Une fonction peut **appeler** une autre fonction. Cette dernière doit être déclarée **avant** celle qui l'appelle.
Par contre, l'imbrication de fonctions n'est pas autorisée en C++, une fonction ne peut pas être déclarée à l'intérieur d'une autre fonction.

*L'expression **void coucou()** est appelé le prototype de la fonction " coucou "*

Exemple à expérimenter:

Exercice VII_3:

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

void carre()       // declaration de la fonction
{
int n, n2;                 // variables locales a carre
cout<<"ENTRER UN NOMBRE: ";
cin>>n;
n2 = n*n;
cout<<"VOICI SON CARRE: "<<n2<<"\n";
}

void main()              // programme principal
{
clrscr();
carre();   // appel de la fonction
cout<<"POUR CONTINUER FRAPPER UNE TOUCHE: ";
getch();
}
```

Conclusion:
Les variables **n** et **n2** ne sont connues que de la fonction carré. Elles sont appelées variables locales. Aucune donnée n'est échangée entre **main()** et la fonction.

*L'expression **void carre()** est le prototype de la fonction " carre "*

Exemple à expérimenter:

Exercice VII_4:

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

void carre()       // declaration de la fonction
{
int n, n2;                 // variables locales a carre
cout<<"ENTRER UN NOMBRE: ";
cin>>n;
n2 = n*n;
cout<<"VOICI SON CARRE: "<<n2<<"\n";
}

void cube()       // declaration de la fonction
{
int n, n3;                 // variables locales a cube
```

```
cout<<"ENTRER UN NOMBRE: ";
cin>>n;
n3 = n*n*n;
cout<<"VOICI SON CUBE: "<<n3<<"\n";
}

void main()             // programme principal
{
char choix;             // variable locale a main()
cout<<"CALCUL DU CARRE TAPER 2\n";
cout<<"CALCUL DU CUBE TAPER 3\n";
cout<<"\nVOTRE CHOIX: ";
cin>>choix;
switch(choix)
        {
        case '2':carre();break;
        case '3':cube();break;
        }
cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE: ";
getch();
}
```

Conclusion:
Les 2 variables locales **n** sont indépendantes l'une de l'autre.
La variable locale **choix** n'est connue que de main().
Aucune donnée n'est échangée entre les fonctions et le programme principal.

*L'expression **void cube()** est le prototype de la fonction " cube "*

Exemple à expérimenter:

Exercice VII_5:

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

int n;          / / variable globale, connue de tout le programme

void carre()    / / declaration de la fonction
{
int n2;   / / variable locale
cout<<"ENTRER UN NOMBRE: ";
cin>>n;
n2 = n*n;
cout<<"VOICI SON CARRE: "<<n2<<"\n";
}
```

```
void cube()       / / declaration de la fonction
{
int n3;  / / variable locale
cout<<"ENTRER UN NOMBRE: ";
cin>>n;
n3 = n*n*n;
cout<<"VOICI SON CUBE: "<<n3<<"\n";
}

void main()             / / programme principal
{
char choix;             / / variable locale a main()
cout<<"CALCUL DU CARRE TAPER 2\n";
cout<<"CALCUL DU CUBE TAPER 3\n";
cout<<"\nVOTRE CHOIX: ";
cin>>choix;
switch(choix)
        {
        case '2':carre();break;
        case '3':cube();break;
        }
cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE: ";
getch();
}
```

Conclusion:
La variable globale **n** est connue de tout le programme (fonctions et programme principal)
La variable locale choix n'est connue que du programme principal.
L'échange d'information entre la fonction et le programme principal se fait via cette variable globale.

Un programme bien construit (lisible, organisé par fonctions, et donc maintenable) maintenablepossède le moins possible de variables globales.

Exercice VII_6:

Un programme contient la déclaration suivante:
int tab[10] = {1,2,4,8,16,32,64,128,256,512}; // variable globale
Ecrire une fonction de prototype **void affiche(void)** qui affiche les éléments du tableau. Mettre en œuvre cette fonction dans le programme principal.

FONCTION RENVOYANT UNE VALEUR AU PROGRAMME

ET SANS PASSAGE D'ARGUMENTS

Dans ce cas, la fonction, après exécution, renvoie une valeur. Le type de cette valeur est déclaré avec la fonction. La valeur retournée est spécifiée à l'aide du mot réservé *return*. Cette valeur peut alors être exploitée par le sous-programme appelant.
Le transfert d'information a donc lieu de la fonction vers le sous-programme appelant.

Exemple à expérimenter:

Exercice VII_7:

```
#include <iostream.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>

int lance_de()  // declaration de la fonction
{
int test;  //variable locale
test = random(6) + 1;        // random(n) retourne une valeur comprise entre
return test;                 // 0 et  n-1
}

void main()
{
int resultat;
randomize();
resultat = lance_de();
     // resultat prend la valeur retournee par le sous-programme
cout<<"Vous avez obtenu le nombre: "<<resultat<<"\n";
cout<<"POUR SORTIR FRAPPER UNE TOUCHE ";
getch();
}
```

Conclusion:

Lors de l'exécution de la fonction " lance_de ", la variable locale **test** est calculée. Sa valeur est exploitée dans le programme principal via la variable **resultat**.

*L'expression **int lance_de()** est le prototype de la fonction " lance_de "*

Exercice VII_8:

Un programme contient la déclaration suivante:
float liste[8] = {1.6,-6,9.67,5.90,345,-23.6,78,34.6}; // variable globale
Ecrire une fonction de prototype **float min()** qui renvoie le minimum de la liste.
Ecrire une fonction de prototype **float max( )** qui renvoie le maximum de la liste.
Les mettre en oeuvre dans le programme principal.
Ne pas utiliser d'autre variable globale que **tab.**

FONCTIONS AVEC PASSAGE D'ARGUMENTS

Ces fonctions utilisent les valeurs de certaines variables du sous-programme les ayant appelé: on passe ces valeurs au moyen d'arguments déclarés avec la fonction.
Le transfert d'information a donc lieu du programme appelant vers la fonction.

Ces fonctions peuvent aussi, si nécessaire, retourner une valeur au sous-programme appelant via le mot *return.*

Exemple à expérimenter:

Exercice VII_9:

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

int carre(int x)  // declaration de la fonction
{               // x est un parametre
int x2;  // variable locale
x2 = x*x;
return x2 ;
}

void main()
{
int n1, n2, res1, res2;                    /* variables locales */
cout<<"ENTRER UN NOMBRE: ";
cin>>n1;
res1 = carre(n1);
cout<<"ENTRER UN AUTRE NOMBRE: ";
cin>>n2;
res2 = carre(n2);
cout<<"VOICI LEURS CARRES: "<<res1<<"  "<< res2;
cout<<"\n\nPOUR SORTIR FRAPPER UNE TOUCHE: ";
getch();  }
```

Conclusion:

x est un paramètre, ou argument. Ce n'est pas une variable du programme.
Sa valeur est donnée via n1 puis n2 par le programme principal.
On dit que l'on a passé le paramètre PAR VALEUR..

On peut ainsi appeler la fonction carre autant de fois que l'on veut avec des variables différentes.

Il peut y avoir autant de paramètre que nécessaire. Ils sont alors séparés par des virgules.
S'il y a plusieurs arguments à passer, il faut respecter la syntaxe suivante:

void fonction1(int x,int y) void fonction2(int a,float b,char c)

Dans l'exercice VII_9, la fonction retourne aussi une valeur au programme principal.
Valeur retournée et paramètre peuvent être de type différent.

*L'expression **int carre(int)** est appelée le prototype réduit de la fonction " carre ".*
*L'expression **int carre(int x)** est appelée le prototype complet de la fonction " carre ".*

Exercice VII_10:

Ecrire une fonction de prototype **int puissance(int a, int b)** qui calcule a^b, a et b sont des entiers (cette fonction n'existe pas en bibliothèque). La mettre en oeuvre dans le programme principal.

PASSAGE DES TABLEAUX ET DES POINTEURS EN ARGUMENT

Exemple à expérimenter:

Exercice VII_11:

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

int somme(int *tx)
{
int i,total=0;
for(i=0;i<20;i++)
        total = total + tx[i];
return total;
}
```

```
void main()
{
int tab[20]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19};
int resultat;
resultat = somme(tab);
cout<<"Somme des elements du tableau : "<<resultat;
cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE ";
getch();
}
```

Conclusion:

La tableau est passé au sous-programme comme un pointeur.
On dit, dans ce cas, que l'on a passé le paramètre PAR ADRESSE.
La notation **tx[i]** peut, bien sur, être remplacée par ***(tx+i)**

Le prototype réduit de la fonction " somme " est ***int somme(int *)***
Le prototype complet de la fonction " somme " est ***int somme(int *tx)***

Exercice VII_12:

tab1 et tab2 sont des variables locales au programme principal.
int tab1[10] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19};
int tab2[10] = {-19,18,-17,16,-15,14,-13,12,-11,10,-9,8,-7,6,-5,4,-3,2,-1,0};
Ecrire une fonction de prototype **void affiche(int *tx)** qui permet d'afficher les 20 nombres suivant un tableau 4x5.
La mettre en oeuvre dans le programme principal pour afficher tab1 et tab2.

RESUME SUR VARIABLES ET FONCTIONS

On a donc vu qu'une variable **globale** est déclarée au début du programme et qu'elle est connue de tout le programme. **Les variables globales sont initialisées à 0 au début de l'exécution du programme, sauf si on les initialise à une autre valeur.**

On a vu aussi qu'une variable **locale** (déclarée au début d'une fonction ou du programme principal) n'est connue que de cette fonction ou du programme principal. Une telle variable locale est encore appelée **automatique.**

Les variables locales ne sont pas initialisées (sauf si on le fait dans le programme) et elles perdent leur valeur à chaque appel à la fonction.

On peut allonger la durée de vie d'une variable locale en la déclarant **static.** Lors d'un nouvel appel à la fonction, la variable garde la valeur obtenue à la fin de l'exécution précédente. Une variable **static** est initialisée à 0 lors du premier appel à la fonction.

Exemple: **int i;** devient **static int i;**

Exemple à expérimenter:

Exercice VII_13:

Quelle sera la valeur finale de n si i est déclarée comme variable static, puis comme variable automatique ?

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

int resultat;   // initialisee a 0

void calcul()
{
static int i;          // initialisee a 0
i++;
cout<<"i= "<<i<<"\n";
resultat = resultat + i;
}

void main()
{
calcul();
cout<<"resultat= "<<resultat<<"\n";
calcul();
cout<<"resultat= "<<resultat<<"\n";
cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE ";
getch();
}
```

Le C++ autorise la déclaration des variables LOCALES au moment où on en a besoin dans la fonction ou dans le programme principal. Si une variable locale est déclarée au début d'un bloc, sa portée est limitée à ce bloc.

Exemple:

```
void main()
{
for ( int i = 0;i<10;i++ )      // la variable i est connue de tout le programme
{                               // principal
        int j;                  // la variable j n'est connue que du bloc
        ...;
        ...;
        }
}
```

Ce exemple est équivalent à:

```
void main()
{
int i;
for ( i = 0;i<10;i++ ) // la variable i est connue de tout le programme
{                      // principal
        int j;         // la variable j n'est connue que du bloc
        ...;
        ...;
        }
}
```

Cette possibilité autorise une gestion plus précise de la mémoire, mais peut nuire à la structuration des programmes.

PASSAGE D'ARGUMENT PAR VALEUR ET PAR REFERENCE

En langage C++, une fonction ne peut pas modifier la valeur d'une variable passée en argument, si cette variable est passée par valeur. Il faut, pour cela, passer le paramètre par référence.

Exemple à expérimenter: Etude d'une fonction permettant d'échanger la valeur de 2 variables:

Exercice VII_14:

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

void ech(int x,int y)
{
int tampon;
tampon = x;
x = y;
y = tampon;
cout<<"\n\nAdresse de x= "<<&x<<"\n";
```

```
cout<<"Adresse de y= "<<&y<<"\n";
}

void main()
{
int a = 5 , b = 8;
cout<<"Adresse de a= "<<&a<<"\n";
cout<<"Adresse de b= "<<&b<<"\n";
cout<<"\n\nValeurs de a et de b avant l'echange:\n";
cout<<"a= "<<a<<"\n";
cout<<"b= "<<b<<"\n";

ech(a,b);

cout<<"\n\nValeurs de a et de b apres l'echange:\n";
cout<<"a= "<<a<<"\n";
cout<<"b= "<<b<<"\n";

cout<<"Pour continuer, frapper une touche ";
getch();
}
```

a et b sont des variables locales au programme principal, passées en argument à la fonction " ech ". La fonction ne peut pas modifier leur valeur.
L'adresse de a est différente de l'adresse de x. Au moment de l'appel à la fonction, la valeur de a est bien recopiée dans la case-mémoire de x. La recopie n'a pas lieu dans l'autre sens lors de la sortie de la fonction.
On peut tenir le même raisonnement à propos de b et de y.

Remplacer maintenant le prototype de la fonction par **void ech(int &x,int &y)** et expérimenter l'exercice.
L'échange a lieu car la même case-mémoire est utilisée pour stocker a et x.
On dit qu'on a passé le paramètre **par référence.**

Le problème ne se pose pas pour les pointeurs et tableaux puisque l'on fournit directement l'adresse du paramètre à la fonction.

*Dans ce cas, le prototype réduit de la fonction est **void ech(int &, int &)** et le prototype complet de la fonction est **void ech(int &x , int &x**).*

QUELQUES EXERCICES

Exercice VII_15:

Saisir les 3 couleurs d'une résistance, afficher sa valeur.
Une fonction de prototype **float conversion(char *couleur)** calcule le nombre associé à chaque couleur. " couleur " est une chaine de caractères.

Exercice VII_16:

Calculer et afficher les racines de $ax^2+bx+c=0$.
- Une fonction de prototype **void saisie(float &aa,float &bb,float &cc)** permet de saisir a,b,c. Ici, le passage par référence est obligatoire puisque la fonction " saisie " modifie les valeurs des arguments.
- Une fonction de prototype **void calcul(float aa,float bb,float cc)** exécute les calculs et affiche les résultats (passage par référence inutile).
- a, b, c sont des variables locales au programme principal.
- Le programme principal se contente d'appeler **saisie(a,b,c) et calcul(a,b,c).**

Exercice VII_17:

Ecrire une fonction de prototype **void saisie(int *tx)** qui saisie des entiers (au maximum 20), le dernier est 13, et les range dans le tableau tx.
Le programme principal appelle **saisie(tab).**

Modifier ensuite la fonction de prototype **void affiche(int *tx)** de l'exercice VII_12 de sorte d'afficher les nombres en tableau 4x5 mais en s'arrêtant au dernier**.** Compléter le programme principal par un appel à **affiche(tab).**

LES CONVERSIONS DE TYPE LORS D'APPEL A FONCTION

Le langage C++, autorise, dans une certaine mesure, le non-respect du type des arguments lors d'un appel à fonction: le compilateur opère alors une conversion de type.

Exemple:
```
double ma_fonction(int u, float f)
{
.....;                    // fonction avec passage de deux paramètres
.....;
}
```

```
void main()
{
char c; int i, j; float r; double r1, r2, r3, r4;
r1 = ma_fonction( i, r );      // appel standard
r2 = ma_fonction( c, r);       // appel correct, c est converti en int
r3 = ma_fonction( i, j);       // appel correct, j est converti en float
r4 = ma_fonction( r, j);       // appel correct, r est converti en int
                               // et j est converti en float
}
```

Exercice VII_18:

Ecrire une fonction **float puissance(float x,int n)** qui renvoie x^n. La mettre en oeuvre en utilisant les propriétés de conversion de type.

LES ARGUMENTS PAR DEFAUT

En C++, on peut préciser la valeur prise par défaut par un argument de fonction. Lors de l'appel à cette fonction, si on omet l'argument, il prendra la valeur indiquée par défaut, dans le cas contraire, cette valeur par défaut est ignorée.

Exemple:

```
void f1(int n = 3)      // par défaut le paramètre n vaut 3
{
....;
}

void f2(int n, float x = 2.35) // par défaut le paramètre x vaut 2.35
{
....;
}

void f3(char c, int n = 3, float x = 2.35)     // par défaut le paramètre n vaut 3
{                                              // et le paramètre x vaut 2.35
....;
}

void main()
{
char a = 0; int i = 2; float r = 5.6;
f1(i);          // l'argument n vaut 2, l'initialisation par défaut est ignorée
f1();           // l'argument n prend la valeur par défaut
f2(i,r);        // les initialisations par défaut sont ignorées
f2(i);          //  le second paramètre prend la valeur par défaut
```

```
        // f2();  interdit
        f3(a, i, r);      //  les initialisations par défaut sont ignorées
        f3(a, i);         //  le troisième paramètre prend la valeur par défaut
        f3(a);            //  le deuxième et la troisième paramètres prennent les valeurs
}               //  par défaut
```

Remarque:

Les arguments, dont la valeur est fournie par défaut, doivent OBLIGATOIREMENT se situer en fin de liste.
La déclaration suivante est interdite: **void f4(char c=2, int n)**

```
{
....;
}
```

Exercice VII_19:

Reprendre l'exercice précédent. Par défaut, la fonction puissance devra fournir x^4.

LA SURDEFINITION DES FONCTIONS

Le C++ autorise la définition de fonctions *différentes* et portant *le même nom.* Dans ce cas, il faut les différencier par le type des arguments.

Exemple à expérimenter:

Exercice VII_20:

```
#include <iostream.h>
#include <conio.h>
void test(int n = 0,float x = 2.5)
{
cout <<"Fonction N°1 :  ";
cout << "n= "<<n<<"  x="<<x<<"\n";
}

void test(float x = 4.1,int n = 2)
{
cout <<"Fonction N°2 :  ";
cout << "n= "<<n<<"  x="<<x<<"\n";
}
```

```
void main()
{
int i = 5; float r = 3.2;
test(i,r);        // fonction N°1
test(r,i);        // fonction N°2
test(i);          // fonction N°1
test(r);// fonction N°2

// les appels suivants, ambigüs, sont rejetés par le compilateur
// test();
// test (i,i);
// test (r,r);
// les inialisations par défaut de x à la valeur 4.1
// et de n à 0 sont inutilisables
getch() ;}
```

Exemple à expérimenter:

Exercice VII_21:

```
#include <iostream.h>
#include <conio.h>
void essai(float x,char c,int n=0)
{cout<<"Fonction N°1: x = "<<x<<" c = "<<c<<" n = "<<n<<"\n";}

void essai(float x,int n)
{cout<<"Fonction N°2: x = "<<x<<" n = "<<n<<"\n";}

void main()
{
char l='z';int u=4;float y = 2.0;
essai(y,l,u);    // fonction N°1
essai(y,l);      // fonction N°1
essai(y,u);      // fonction N°2
essai(u,u);      // fonction N°2
essai(u,l);      // fonction N°1
// essai(y,y); rejet par le complateur
essai(y,y,u);    // fonction N°1
getch() ;}
```

Exercice VII_22:

Ecrire une fonction **float puissance (float x, float y)** qui retourne x^y (et qui utilise la fonction " **pow "** de la bibliothèque mathématique **math.h**).

Ecrire *une autre* fonction **float puissance(float x, int n)** qui retourne x^n (et qui est écrite en utilisant l'algorithme de l'exercice précédent.
Les mettre en oeuvre dans le programme principal, en utilisant la propriété de surdéfinition.

.

CORRIGE DES EXERCICES

Exercice VII_6:

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

tab[10]={1,2,4,8,16,32,64,128,256,512};

void affiche()
{
int i;
cout<<"VOICI LES ELEMENTS DU TABLEAU ET LEURS ADRESSES:\n\n";
for(i=0;i<10;i++)
        cout<<"ELEMENT Numero  "<<i<<": "<<tab[i]<<"  Son adresse
:"<<(tab+i)<<"\n";
}

void main()
{
affiche();
cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE: ";
getch();
}
```

Exercice VII_8:

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

float liste[8] = {1.6,-6,9.67,5.90,345,-23.6,78,34.6};

float max()
{
float M;
int i;
M = *liste;
for(i=0;i<8;i++)
        if(*(liste+i)>M)
```

```
                M = *(liste+i);
return(M);
}

float min()
{
float m;
int i;
m = *liste;
for(i=0;i<8;i++)
        if(*(liste+i)<m)
                m = *(liste+i);
return(m);
}

void main()
{
float resultat_min, resultat_max;
resultat_max = max();
resultat_min = min();
cout<<"LE MAXIMUM VAUT: "<<resultat_max<<"\n";
cout<<"LE MINIMUM VAUT: "<<resultat_min<<"\n";
cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE";
getch();
}
```

Exercice VII_10:

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

int puissance(int x,int y)
{
int i,p=1;
for(i=1;i<=y;i++) p=x*p;
return(p);
}

void main()
{
int a,b,res;
cout<<"\nENTRER A: ";cin>>a;
cout<<"\nENTRER B: ";cin>>b;
res = puissance(a,b);
cout<<"\nA PUISS B = "<<res;
cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE ";
```

```
getch();
}
```

Exercice VII_12:

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

void affiche(int *tx)
{
int i;
for(i=0;i<20;i++)
        if((i+1)%5==0) cout<<tx[i]<<"\n" ;
        else cout<<tx[i]<<"\t";
cout<<"\n\n";
}

void main()
{
int tab1[20]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19};
int tab2[20]={-19,18,-17,16,-15,14,-13,12,-11,10,-9,8,-7,6,-5,4,-3,2,-1,0};
affiche(tab1);
affiche(tab2);
cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE ";
getch();
}
```

Exercice VII_15:

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

float conversion(char *couleur)
{
float x=10;
couleur = strupr(couleur);  // convertit en majuscules
// stricmp permet d'eviter l'utilisation de strupr
if (strcmp("NOIR",couleur)==0) x=0;
else if (strcmp("MARRON",couleur)==0) x=1;
else if (strcmp("ROUGE",couleur)==0) x=2;
else if (strcmp("ORANGE",couleur)==0) x=3;
else if (strcmp("JAUNE",couleur)==0) x=4;
else if (strcmp("VERT",couleur)==0) x=5;
else if (strcmp("BLEU",couleur)==0) x=6;
```

```
return(x); // x permet d'ajouter  un controle d'erreur
}

void main()
{
float r,c1,c2,c3;
char *coul1,*coul2,*coul3;
coul1 = new char[8];
coul2 = new char[8];
coul3 = new char[8];
cout<<"\nENTRER LES 3 COULEURS DE LA RESISTANCE:\n";
cout<<"COULEUR1: ";cin>>coul1;
cout<<"COULEUR2: ";cin>>coul2;
cout<<"COULEUR3: ";cin>>coul3;
c1=conversion(coul1);
c2=conversion(coul2);
c3=conversion(coul3);

if( (c1==10) || (c2==10) || (c3==10) ) cout<<"Erreur\n";
else
{
r = (c1*10 + c2)*pow(10,c3);
if(r<1000) cout<<"\nVALEUR DE R: "<<r<<" OHM\n";
if((r>=1000)&&(r<999999))
        {
        r=r/1000;
        cout<<"\nVALEUR DE R: "<<(int)r<<" KOHM\n";
        }
if(r>=999999)
        {
        r=r/1e6;
        cout<<"\nVALEUR DE R: "<<(int)r<<" MOHM\n";
        }
}
delete coul1;
delete coul2;
delete coul3;
cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE ";
getch();
}
```

Exercice VII_16:

```
#include <iostream.h>
#include <math.h>
#include <conio.h>

void saisie(float &aa,float &bb,float &cc) // par reference
                        // car fonction de saisie
{
cout<<"\nENTRER A: ";cin>>aa;
cout<<"\nENTRER B: ";cin>>bb;
cout<<"\nENTRER C: ";cin>>cc;
cout<<"\n";
}

void calcul(float aa,float bb,float cc)
{
float delta,x1,x2;
cout<<"\nA= "<<aa<<" B= "<<bb<<" C= "<<cc<<"\n";
delta = bb*bb-4*cc*aa;
cout<<"\nDELTA = "<<delta<<"\n";
if (delta<0) cout<<"\nPAS DE SOLUTION";
else if (delta == 0)
{
x1=-(bb/aa/2);
cout<<"\UNE SOLUTION: X= "<<x1<<"\n";
}
else
{
x1=(-bb+sqrt(delta))/2/aa;
x2=(-bb-sqrt(delta))/2/aa;
cout<<"\DEUX SOLUTIONS: X1 = "<<x1<<"  X2 = "<<x2<<"\n";
}
}

void main()
{
float a,b,c;
saisie(a,b,c);
calcul(a,b,c);
cout<<"\n\nPOUR SORTIR FRAPPER UNE TOUCHE ";
getch();
}
```

Exercice VII_17:

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

void saisie(int *tx)
{
int i=0;
cout<<"SAISIE DES NOMBRES SEPARES PAR RETURN (dernier =13)\n";
do
{
cout<<"NOMBRE: ";
cin>>*(tx+i);        // ou bien tx[i-1]
i++;
}
while(*(tx-1+i)!=13);       // ou bien tx[i-1]
}

void affiche(int *tx)
{
int i;
cout<<"\n";
for(i=0;*(tx+i-1)!=13;i++)
if((i+1)%5==0)cout<<tx[i]<<"\n";
else cout<<tx[i]<<"\t";
}

void main()
{
int tab[20];
saisie(tab);
affiche(tab);
cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE ";
getch();
}
```

Exercice VII_18:

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

float puissance(float x,int n)
{
float resultat=1;
for(int i=1;i<=n;i++)resultat = resultat * x;
return resultat;
}
```

```
void main()
{
char c=5;int i=10,j=6; float r=2.456,r1,r2,r3,r4,r5;
r1 = puissance(r,j);
r2 = puissance(r,c);
r3 = puissance(j,i);
r4 = puissance(j,r);
r5 = puissance(0,4);
cout << "r1 = " <<r1<<"\n";
cout << "r2 = " <<r2<<"\n";
cout << "r3 = " <<r3<<"\n";
cout << "r4 = " <<r4<<"\n";
cout << "r5 = " <<r5<<"\n";
getch();
}
```

Exercice VII_19:

```
#include <iostream.h>
#include <conio.h>
float puissance(float x,int n=4)
{
float resultat=1;
for(int i=1;i<=n;i++)resultat = resultat * x;
return resultat;
}

void main()
{
int j=6; float r=2.456,r1,r2,r3,r4,r5;
r1 = puissance(r,j);
r2 = puissance(r);
r3 = puissance(1.4,j);
r4 = puissance(1.4);
cout << "r1 = " <<r1<<"\n";
cout << "r2 = " <<r2<<"\n";
cout << "r3 = " <<r3<<"\n";
cout << "r4 = " <<r4<<"\n";
getch();
}
```

Exercice VII_22:

```
#include <iostream.h>
#include <conio.h>
#include <math.h>

float puissance(float x,int n)
{
float resultat = 1;
for(int i=0;i<n;i++)resultat = resultat * x;
return resultat;
}

float puissance(float x,float y)
{float resultat;
resultat = pow(x,y);
return resultat;
}

void main()
{
int b1=4;
float a = 2.0, b2 = 0.5,r1,r2;
r1 = puissance(a,b1);
r2 = puissance(a,b2);
cout<<"r1= "<<r1<<"  r2= "<<r2<<"\n";
cout<<"Frapper une touche";
getch();
}
```