COURS et TP DE LANGAGE C++

Chapitre 1

Eléments de langage C++

Joëlle MAILLEFERT

joelle.maillefert@iut-cachan.u-psud.fr

IUT de CACHAN

Département GEII 2

CHAPITRE 1

ELEMENTS DE LANGAGE C++

Les exercices ont été testés avec les outils BORLAND C++ BUILDER (toute version) en mode « console » et BC5. Le corrigé des exercices et le listing de ces programmes se trouvent à la fin de chaque chapitre et sont téléchargeables.

Pour avancer un peu plus vite et aborder l'essentiel de la Programmation Orientée Objet (P.O.O.), on pourra étudier les chapitres et paragraphes marqués de ***, dans un deuxième temps.

INTRODUCTION

Le langage C++ est un langage évolué et structuré. C'est en ce sens une évolution du langage C.
Il possède en outre les fonctionnalités de la programmation orienté objet.
Le langage C++ se trouve à la frontière entre le langage C, non objet, et le langage JAVA conçu d'emblée en orienté objet.

On trouve sur le marché un grand nombre de compilateurs C++ destinés à différents microprocesseurs ou microcontrôleurs.

Le langage C++ possède assez peu d'instructions, il fait par contre appel à des bibliothèques, fournies en plus ou moins grand nombre avec le compilateur.

exemples: math.h : bibliothèque de fonctions mathématiques
iostream.h : bibliothèque d'entrées/sorties standard
complex.h : bibliothèque contenant la classe des nombres complexes.

On ne saurait développer un programme en C++ sans se munir de la documentation concernant ces bibliothèques.

ETAPES PERMETTANT L'EDITION, LA MISE AU POINT,

L'EXECUTION D'UN PROGRAMME

*1- **Edition du programme source***, à l'aide d'un éditeur (traitement de textes). Le nom du fichier contient l'extension .CPP, exemple: EXI_1.CPP (menu « edit »).

*2- **Compilation du programme source,*** c'est à dire création des codes machine destinés au microprocesseur utilisé. Le compilateur indique les erreurs de syntaxe mais ignore les fonctions-bibliothèque appelées par le programme.
Le compilateur génère un fichier binaire, non éditable en mode « texte », appelé fichier objet: EXI_1.OBJ (commande « compile »).

*3- **Editions de liens***: Le code machine des fonctions-bibliothèque est chargé, création d'un fichier binaire, non éditable en mode texte, appelé fichier exécutable: EXI_1.EXE (commande « build all »).

*4- **Exécution du programme*** (commande « Run » ou « flèche jaune »).

Les compilateurs permettent en général de construire des programmes composés de plusieurs fichiers sources, d'ajouter à un programme des unités déjà compilées. On dit alors que l'on travaille par gestion de projet.

Exercice I-1: Editer (EXI_1.CPP), compiler et exécuter le programme suivant:

```
#include <iostream.h>  // sorties standards
#include <conio.h>     // les commentaires s'écrivent derrière 2 barres
    void main()
   {
      cout<<"BONJOUR";//affichage d'un message sur l'écran
      cout<<" Belle journée!!";//affichage d'un autre message sur l'écran
      cout<<"Pour continuer frapper une touche...";
   // Attente d'une saisie clavier pour voir l'écran d'exécution
      getch();
   }
```

Le langage C++ distingue les minuscules, des majuscules. Les mots réservés du langage C++ doivent être écrits **en minuscules.**

On a introduit dans ce programme la notion d'interface homme/machine (IHM).
- L'utilisateur visualise une information sur l'écran,
- L'utilisateur, par une action sur le clavier, fournit une information au programme.

Les instructions sont exécutées **séquentiellement**, c'est à dire les unes après les autres. L'ordre dans lequel elles sont écrites a donc une grande importance.

Echanger les 2 premières instructions, puis exécuter le programme.

Modifier maintenant le programme comme ci-dessous, puis le tester :

```
//les commentaires s'écrivent derrière 2 barres obliques

#include <iostream.h>  //sorties standard
#include <conio.h>

    void main()
   {
      int a, b, calcul ;  //déclaration de 3 variables
      cout<<"BONJOUR";//affichage d'un message sur l'écran
      a = 10 ; // affectation
      b = 50 ; // affectation
      calcul = (a + b)*2 ;  //
      cout <<" Affichage de a : "<< a<<"\n";
      cout <<" Affichage de b : "<< b<<"\n";
      cout <<" Voici le résultat : "<< calcul<<"\n";
      cout<<"Pour continuer frapper une touche...";
      getch();  // Attente d'une saisie clavier
   }
```

Dans ce programme, on introduit 3 nouveaux concepts :
- La notion de déclaration de variables : les variables sont les données que manipulera le programme lors de son exécution. Ces variables sont rangées dans la mémoire vive de l'ordinateur. Elles peuvent être déclarées au moment où on en a besoin dans le programme. Pour une meilleure lisibilité, il est conseillé de les déclarer au début (sauf peut-être pour des variables créées par commodité et qui ne servent que très localement dans le programme).
- La notion d'affectation, symbolisée par le signe =. La source de l'information est à droite du signe =, la destination à gauche.

a = 10; signifie « a prend la valeur 10 »

s = a + b; signifie « s prend la valeur a + b »

s = s + 5; signifie « la nouvelle valeur de s est égale à l'ancienne + 5 »

- La notion d'opération. Un programme informatique est exécuté séquentiellement, c'est à dire une instruction après l'autre. Lorsque l'instruction **s = a + b** est exécutée, **a** possède la valeur 10, et **b** possède la valeur 50.

LES DIFFERENTS TYPES DE VARIABLES

1- Les entiers

Le langage C++ distingue plusieurs types d'entiers:

TYPE	DESCRIPTION	TAILLE MEMOIRE
int	entier standard signé	4 octets: $-2^{31} \leq n \leq 2^{31}-1$
unsigned int	entier positif	4 octets: $0 \leq n \leq 2^{32}$
short	entier court signé	2 octets: $-2^{15} \leq n \leq 2^{15}-1$
unsigned short	entier court non signé	2 octets: $0 \leq n \leq 2^{16}$
char	caractère signé	1 octet : $-2^{7} \leq n \leq 2^{7}-1$
unsigned char	caractère non signé	1 octet : $0 \leq n \leq 2^{8}$

Numération:

- En décimal les nombres s'écrivent tels que,
- En hexadécimal ils sont précédés de 0x.

exemple: 127 en décimal s'écrit 0x7f en hexadécimal.

Remarque: En langage C++, le type **char** possède une fonction de changement de type vers un entier:
- Un caractère peut voir son type automatiquement transformé vers un entier de 8 bits
- Il est interprété comme un caractère alphanumérique du clavier.

Exemples:

Les caractères alphanumériques s'écrivent entre ‘ ‘

Le **caractère** 'b' a pour valeur 98.

Le **caractère** 22 a pour valeur 22.

Le **caractère** 127 a pour valeur 127.

Le **caractère** 257 a pour valeur 1 (ce nombre s'écrit sur 9 bits, le bit de poids fort est perdu).

Quelques constantes caractères:

CARACTERE		VALEUR (code ASCII)	NOM ASCII
'\n'	interligne	0x0a	LF
'\t'	tabulation horizontale	0x09	HT
'\v'	tabulation verticale	0x0b	VT
'\r'	retour chariot	0x0d	CR
'\f'	saut de page	0x0c	FF
'\\'	backslash	0x5c	\
'\''	cote	0x2c	'
'\"'	guillemets	0x22	"

Modifier ainsi le programme et le tester :

```
#include <iostream.h>  // sorties standard
#include <conio.h>     // les commentaires s'écrivent derrière 2 barres

    void main()
   {
      int a, b, calcul ;  // déclaration de 3 variables
      char u ,v ;
      cout<<"BONJOUR"; // affichage d'un message sur l'écran
      a = 10 ; // affectation
      b = 50 ; // affectation
      u = 67 ;
      v = 'A' ;
      calcul = (a + b)*2 ; //affectation et opérations
      cout <<" Affichage de a : "<< a<<"\n";
      cout <<" Affichage de b : "<< b<<"\n";
      cout <<" Voici le résultat : "<< calcul<<"\n";
      cout <<" Affichage de u :"<< u <<"\n";
      cout <<" Affichage de v :"<< v <<"\n" ;
      cout<<"Pour continuer frapper une touche...";
      getch(); // Attente d'une saisie clavier
   }
```

2- Les réels

Un réel est composé :
- d'un signe,
- d'une mantisse,
- d'un exposant,

Un nombre de bits est réservé en mémoire pour chaque élément.

Le langage C++ distingue 2 types de réels:

TYPE	DESCRIPTION	TAILLE MEMOIRE
float	réel standard	4 octets
double	réel double précision	8 octets

LES INITIALISATIONS

Le langage C++ permet l'initialisation des variables dès leurs déclarations:

char c; est équivalent à char c = 'A';
c = 'A';

int i; est équivalent à int i = 50;
i = 50;

Cette règle s'applique à tous les nombres, char, int, float ... Pour améliorer la lisibilité des programmes et leur efficacité, il est conseillé de l'utiliser.

SORTIES DE NOMBRES OU DE TEXTE A L'ECRAN

L'OPERATEUR COUT

Ce n'est pas une instruction du langage C++, mais une fonction de la bibliothèque iostream.h.

Exemple: affichage d'un texte:

```
cout <<"BONJOUR";        // pas de retour à la ligne du curseur après l'affichage
cout <<"BONJOUR\n";      // affichage du texte, puis retour à la ligne du curseur
```

Exercice I-2: Tester le programme suivant et conclure.

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

    void main()
   {
      cout<<"BONJOUR " ;
      cout <<"IL FAIT BEAU\n";
      cout <<"BONNES VACANCES";
      cout <<"Pour continuer frapper une touche...";
      getch();  // Attente d'une saisie clavier
   }
```

Exercice I-3: Affichage d'une variable de type **int** ou **float:**

Tester le programme suivant et conclure.

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

    void main()
   {
      int u = 1000 ;
      float s = 45.78 ;
      cout <<"Voici u (en base 10) :  " << u << "\n";
      cout <<"Voici u (en hexa) : "<< hex << u <<"\n";
      cout <<"Voici s : "<< s << "\n";
      cout <<"Pour continuer frapper une touche...";
      getch();  // Attente d'une saisie clavier
   }
```

Affichage multiple: modifier le programme précédent ainsi, et conclure.

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

    void main()
   {
      int u;
      float s;
      u = 1000;
      s = 45.78;
      cout <<"Voici u (base 10) : "<< u << "\nVoici s : " << s << "\n";
      cout <<"Pour continuer frapper une touche...";
      getch();  // Attente d'une saisie clavier
   }
```

Exercice I-4:

a et b sont des entiers, a = -21430 b = 4782, calculer et afficher a+b, a-b, a*b, a/b, a%b en soignant l'interface homme/machine.
Indication: a/b donne le quotient de la division, a%b donne le reste de la division.

Exercice I-5: Affichage d'une variable de type **char :** tester le programme ci-dessous et conclure.

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

    void main()
   {
      char u,v,w;
      int i;
      u = 'A';
      v = 67;
      w = 0x45;
      cout<<"Voici u : "<< u << "\n";
      cout<<"Voici v : "<< v <<"\n";
      cout<<"Voici w : "<< w <<"\n";
      i = u; // conversion automatique de type
      // pour obtenir le code ascii de la lettre A en base 10
      cout<<"Voici i : "<< i << "\n";
      // pour obtenir le code ascii de la lettre A en hexadécimal
      cout<<"Voici i : "<< hex << i << "\n";
      cout<<"Pour continuer frapper une touche...";
      getch();  // Attente d'une saisie clavier
   }
```

Exercice I-6:

Pour votre compilateur C++, la taille des entiers est de 32 bits;
Que va-t-il se passer, à l'affichage, lors de l'exécution du programme suivant ?

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

    void main()
   {
      int a = 12345000, b = 60000000, somme;
      somme=a*b;
      cout<<"a*b = "<<somme<<"\n";

      cout <<"Pour continuer frapper une touche...";
      getch();  /* Attente d'une saisie clavier */
   }
```

Exercice I-7:

a et b sont des réels, a = -21,43 b = 4,782, calculer et afficher a+b, a-b, a*b, a/b, en soignant l'interface homme/machine.

LES OPERATEURS

Opérateurs arithmétiques sur les réels: + - * / avec la hiérarchie habituelle.

Opérateurs arithmétiques sur les entiers: + - * / (quotient de la division) % (reste de la division) avec la hiérarchie habituelle.

Exemple particulier: char c, d; c = 'G'; d = c+'a'-'A';
Les caractères sont des entiers sur 8 bits, on peut donc effectuer des opérations. Sur cet exemple, on transforme la lettre majuscule G en la lettre minuscule g.

Opérateurs logiques sur les entiers:

& ET | OU ^ OU EXCLUSIF ~ COMPLEMENT A UN
« DECALAGE A GAUCHE
» DECALAGE A DROITE.

Exemples: p = n « 3; // p est égale à n décalé de 3 bits à gauche
p = n » 3; // p est égale à n décalé de 3 bits à droite

L'opérateur **sizeof(type)** renvoie le nombre d'octets réservés en mémoire pour chaque type d'objet.
Exemple: n = sizeof(char); /* n vaut 1 */

Exercice I-8: n est un entier (n = 0x1234567a), p est un entier (p = 4). Ecrire un programme qui met à 0 les p bits de poids faibles de n.

Exercice I-9: quels nombres va renvoyer le programme suivant ?

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

    void main()
   {
      cout<<"TAILLE D'UN CARACTERE : "<<sizeof(char)<< "\n";
      cout<<"TAILLE D'UN ENTIER : "   <<sizeof(int)<< "\n";
      cout<<"TAILLE D'UN REEL : "     <<sizeof(float)<< "\n";
      cout<<"TAILLE D'UN DOUBLE : "   <<sizeof(double)<< "\n";
      cout <<"Pour continuer frapper une touche...";
      getch();  // Attente d'une saisie clavier
   }
```

INCREMENTATION - DECREMENTATION

Le langage C++ autorise des écritures simplifiées pour l'incrémentation et la décrémentation de variables de type entier (int, char, long)

i = i+1; est équivalent à **i++;**

i = i-1; est équivalent à **i--;**

OPERATEURS COMBINES

Le langage C++ autorise des écritures simplifiées lorsqu'une même variable est utilisée de chaque côté du signe = d'une affectation. Ces écritures sont à éviter lorsque l'on débute l'étude du langage C++ car elles nuisent à la lisibilité du programme.

a = a+b;	est équivalent à	a+= b;
a = a-b;	est équivalent à	a-= b;
a = a & b;	est équivalent à	a&= b;

LES DECLARATIONS DE CONSTANTES

Le langage C++ autorise 2 méthodes pour définir des constantes.

1ere méthode: **déclaration** d'une variable, dont la valeur sera constante pour toute la portée de la fonction main.

Exemple :

```
 void main()
{
   const float PI = 3.14159;
   float perimetre, rayon = 8.7;
   perimetre = 2*rayon*PI;
// ...
}
```

Dans ce cas, le compilateur réserve de la place en mémoire (ici 4 octets), pour la variable pi, on ne peut changer la valeur. On peut associer un modificateur« const » à tous les types.

2eme méthode: **définition d'un symbole** à l'aide de la directive de compilation **#define.**

Exemple:

```
#define PI = 3.14159;

 void main()
{
   float perimetre,rayon = 8.7;
   perimetre = 2*rayon*PI;
// ....
}
```

Le compilateur ne réserve pas de place en mémoire, on définit ainsi une équivalence « lexicale ».
Les constantes déclarées par #define s'écrivent traditionnellement en majuscules, mais ce n'est pas une obligation.

LES CONVERSIONS DE TYPES

Le langage C++ permet d'effectuer automatiquement des conversions de type sur les scalaires:

Exemple et exercice I-11:

```
 void main()
{
   char c=0x56,d=25,e;
   int i=0x1234,j;
   float r=678.9,s;
   j = c; // j vaut 0x0056, utilisé précédemment  pour afficher
          // le code ASCII d'un caractère
   j = r; // j vaut 678
   s = d; // s vaut 25.0
   e = i; // e vaut 0x34
}
```

Une conversion de type float --> int ou char peut-être *dégradante.*

Une conversion de type int ou char --> float est dite *non dégradante.*

CORRIGE DES EXERCICES

Exercice I-4:

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

    void main()
   {
      int a,b;
      a= -21430;
      b= 4782;
      cout<<"A + B = "<< a+b <<"\n";
      cout<<"A - B = "<< a-b <<"\n";
      cout<<"A x B = "<< a*b <<"\n";
      cout<<"A sur B = "<< a/b <<"\n" ;
      cout<<"A mod B = "<< a%b <<"\n";
      cout<<"Pour continuer frapper une touche...";
      getch();  // Attente d'une saisie clavier
   }
```

Exercice I-7:

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

    void main()
   {
      float a,b;

      a= -21430;
      b= 4782;

      cout<<"A + B = "<< a+b <<"\n";
      cout<<"A - B = "<< a-b <<"\n";
      cout<<"A x B = "<< a*b <<"\n";
      cout<<"A sur B = "<< a/b <<"\n" ;

      cout<<"Pour continuer frapper une touche...";
      getch();  // Attente d'une saisie clavier
   }
```

Exercice I-8:

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

    void main()
   {
      int n,p,masque;

      n= 0x1234567a;
      p = 4;

      cout<<"valeur de n avant modification:"<< hex << n <<"\n";
      n = n >> p;
      n = n << p;

      cout<<"n modifié vaut:"<< hex << n <<"\n";
      cout <<"Pour continuer frapper une touche...";

      getch();  // Attente d'une saisie clavier
   }
```

Exercice I-9:

Avec le compilateur C++ utilisé :

- sizeof(char) vaut 1
- sizeof(int) vaut 4
- sizeof(float) vaut 4
- sizeof(double) vaut 8.

COURS et TP DE LANGAGE C++

Chapitre 2

Saisie de nombres et de caractères au clavier

Joëlle MAILLEFERT

joelle.maillefert@iut-cachan.u-psud.fr

IUT de CACHAN

Département GEII 2

CHAPITRE 2

SAISIE DE NOMBRES ET DE CARACTERES AU CLAVIER

LA FONCTION GETCH

La fonction getch, appartenant à la bibliothèque conio.h permet la saisie clavier d'un caractère alphanumérique, **sans écho écran**. La saisie s'arrête dès que le caractère a été frappé.

La fonction getch n'est pas définie dans la norme ANSI mais elle existe dans les bibliothèques des compilateurs.

On peut utiliser getch de deux façons:

- sans retour de variable au programme:

```
Exemple:    cout<<"POUR CONTINUER FRAPPER UNE TOUCHE ";
            getch();
```

- avec retour de variable au programme:

```
Exemple:    char alpha;
            cout<<"ENTRER UN CARACTERE (ATTENTION PAS DE
RETURN)";
            alpha = getch();
            cout<<"\nVOICI CE CARACTERE: "<<alpha;
```

Les parenthèses vides de getch() signifient qu'aucun paramètre n'est passé à cette fonction par le programme appelant.

L'OPERATEUR CIN

L'opérateur **cin, spécifique à C++,** appartient à la bibliothèque iostream.h, et permet la saisie à partir du clavier de n'importe quel type de variable (l'affichage prend en compte le type de la variable).
La saisie s'arrête avec "RETURN" (c'est à dire LF), les éléments saisis s'affichent à l'écran (**saisie avec écho écran**).

Tous les éléments saisis après un **caractère d'espacement** (espace, tabulation) sont ignorés.

Exemples:
```
char alpha;
int i;
float r;
cin >>alpha;        // saisie d'un caractère
cin >>i;            // saisie d'un nombre entier en décimal
cin >>r;            // saisie d'un nombre réel
```

Remarque: Si l'utilisateur ne respecte pas le type de la variable, aucune erreur n'est générée.
Le programme peut se comporter de plusieurs façons :

Exemples:
```
int u;
cin >> u;
```
Si l'utilisateur saisi un caractère non numérique, sa saisie est ignorée.

```
char c;
cin >> c;
```
Si l'utilisateur saisi par exemple 68, le caractère '6' sera affecté à la variable c.
Conséquence : pour une interface homme machine (IHM) d'un produit fini, ne jamais utiliser « cin ».

Exercice II_1:

Saisir un caractère au clavier, afficher son code ASCII à l'écran. Soigner l'affichage.

Exercice II_2:

Dans une élection, I est le nombre d'inscrits, V le nombre de votants,
P = 100V/I le pourcentage de votants, M = V/2 le nombre de voix pour obtenir la majorité.
Ecrire un programme qui demande à l'utilisateur de saisir I et V, puis calcule et affiche P et M.

Exercice II_3:

Saisir 3 réels, calculer et afficher leur moyenne.

CORRIGE DES EXERCICES

Exercice II_1:

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

    void main()
   {
      char c;
      int u;
      cout<<"ENTRER UN CARACTERE : ";
      cin >> c;
      u = c;    //conversion automatique de type
      cout<<"VOICI SON CODE ASCII : "<< u << "\n";
      cout<<"Pour continuer frapper une touche...";
      getch();
   }
```

Exercice II_2:

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

    void main()
   {
      int I,V,M,P;
      cout<<"Entrer le nombre d'inscrits : ";
      cin>>I;
      cout<<"Entrer le nombre de votants : ";
      cin>>V;
      P = V*100/I;
      M = V/2 + 1; // Division entière
      cout<<"Participation : "<<P<<"% - Majorité: ";
      cout<<M<<" bulletins\n";
      cout<<"POUR CONTINUER FRAPPER UNE TOUCHE ";
      getch();
   }
```

Exercice II_3:

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

    void main()
   {
      float r1, r2, r3, moy;

      cout<<"ENTRER UN NOMBRE REEL : ";
      cin >> r1;

      cout<<"ENTRER UN NOMBRE REEL : ";
      cin >> r2;

      cout<<"ENTRER UN NOMBRE REEL : ";
      cin >> r3;

      moy = (r1 + r2 + r3) / 3;

      cout<<"MOYENNE DE CES 3 NOMBRES : "<<moy<<"\n";
      cout<<"Pour continuer frapper une touche ...";

      getch();
   }
```

COURS et TP DE LANGAGE C++

Chapitre 3

Les tests et les boucles

Joëlle MAILLEFERT

joelle.maillefert@iut-cachan.u-psud.fr

IUT de CACHAN

Département GEII 2

CHAPITRE 3

LES TESTS ET LES BOUCLES

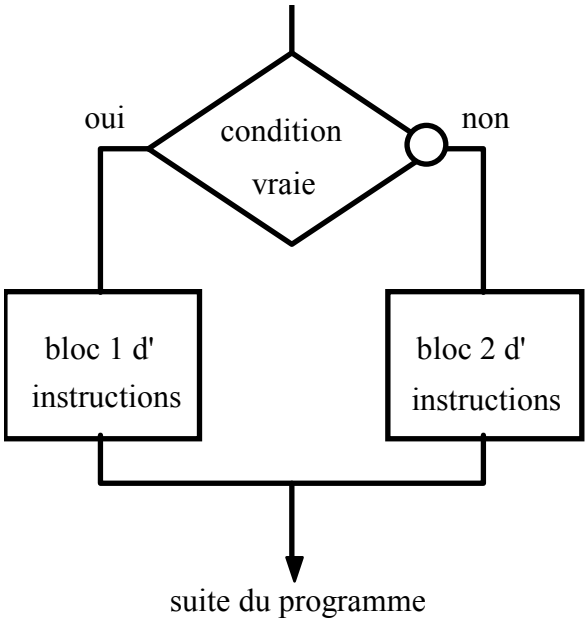
Un programme écrit en C++ s'exécute séquentiellement, c'est à dire instruction après instruction.
Ce chapitre explique comment, dans un programme, on pourra :

- ne pas exécuter une partie des instructions, c'est à dire faire un saut dans le programme,
- revenir en arrière, pour exécuter plusieurs fois de suite la même partie d'un programme.

L'INSTRUCTION SI ... ALORS ... SINON ...

Il s'agit de l'instruction: **si (condition vraie)**
alors {BLOC 1 D'INSTRUCTIONS}
sinon {BLOC 2 D'INSTRUCTIONS}

Organigramme:



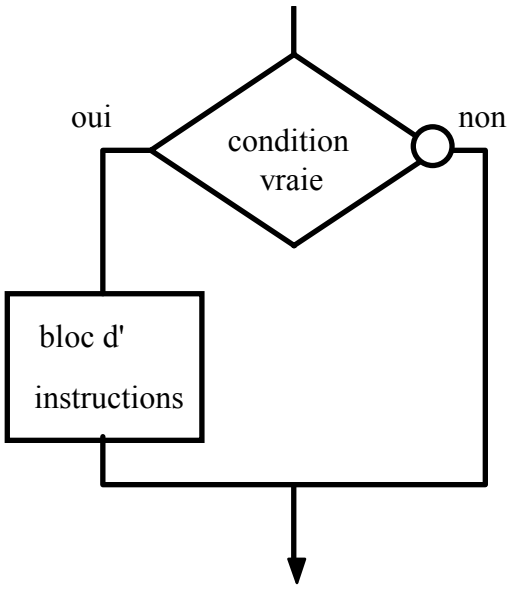
Syntaxe en C:

```
if (condition)
        {
        .....;          // bloc 1 d'instructions
        .....;
        .....;
        }
        else
```

```
{
.....;                    // bloc 2 d'instructions
.....;
.....;
}
suite du programme ...
```

Si la condition est vraie, seul le bloc1 d'instructions est exécuté, si elle est fausse, seul le bloc2 est exécuté.
Dans tous les cas, la suite du programme sera exécutée.

Le bloc "sinon" est optionnel: **si (condition)**
alors {BLOC D'INSTRUCTIONS}



Syntaxe en C:

```
if (condition)
        {
        .....;                    // bloc d'instructions
        .....;
        .....;
        }
suite du programme...
```

Si la condition est vraie, seul le bloc1 d'instructions est exécuté, si elle est fausse, on passe directement à la suite du programme.

Remarque: les {} ne sont pas nécessaires lorsque les blocs ne comportent qu'une seule instruction.

LES OPERATEURS LOGIQUES

condition d'égalité: **if (a==b)** **" si a est égal à b "**

condition de non égalité: **if (a!=b)** **" si a est différent de b "**

conditions de relation d'ordre: **if (a<b)** **if (a<=b)** **if (a>b)** **if (a>=b)**

plusieurs conditions devant être vraies simultanément, ET LOGIQUE:
if ((expression1) && (expression2)) **" si l'expression1 ET l'expression2 sont vraies "**

une condition devant être vraie parmi plusieurs, OU LOGIQUE
if ((expression1) || (expression2)) **" si l'expression1 OU l'expression2 est vraie "**

condition fausse **if (!(expression1))** **" si l'expression1 est fausse "**

Toutes les combinaisons sont possibles entre ces tests.

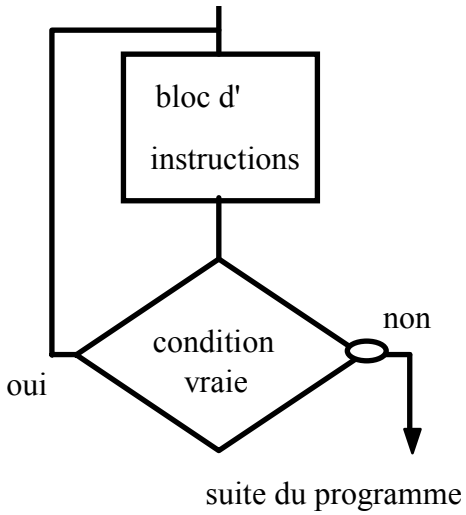
Exercice III-1: L'utilisateur saisit un caractère, le programme teste s'il s'agit d'une lettre majuscule, si oui il renvoie cette lettre en minuscule, sinon il renvoie un message d'erreur.

Exercice III-2: Dans une élection, I est le nombre d'inscrits, V le nombre de votants, Q le quorum, P = 100V/I le pourcentage de votants, M = V/2 + 1 le nombre de voix pour obtenir la majorité absolue.
Le quorum est le nombre minimum de votants pour que le vote soit déclaré valable.
Ecrire un programme qui
1- demande à l'utilisateur de saisir I, Q et V,
2- teste si le quorum est atteint,
3- si oui calcule et affiche P, M, sinon affiche un message d'avertissement.

L'INSTRUCTION REPETER ... TANT QUE ...

Il s'agit de l'instruction: **exécuter {BLOC D'INSTRUCTIONS}**
tant que (condition vraie)

Organigramme:



Syntaxe en C:

```
do
        {
        .....;          // bloc d'instructions
        .....;
        .....;
        }
while (condition);
suite du programme ...
```

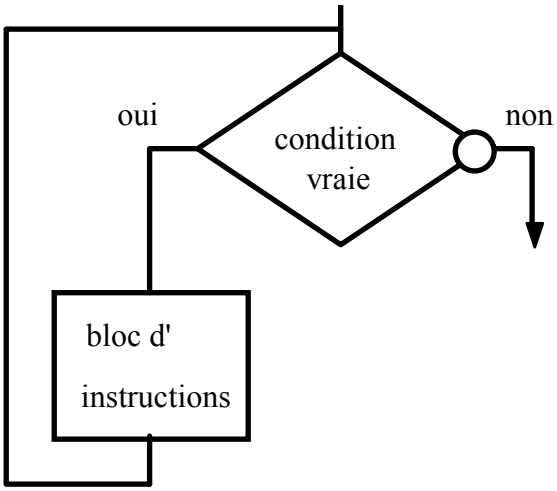
Le test se faisant **après,** le bloc est exécuté au moins une fois.

Remarque: les { } ne sont pas nécessaires lorsque le bloc ne comporte qu'une seule instruction.

LA BOUCLE TANT QUE ... FAIRE ...

Il s'agit de l'instruction: **tant que (condition vraie)**
Exécuter {BLOC D'INSTRUCTIONS}

Organigramme:



Syntaxe en C:

```
while (condition)
        {
        .....;              // bloc d'instructions
        .....;
        .....;
        }
suite du programme ...
```

Le test se fait **d'abord,** le bloc d'instructions n'est pas forcément exécuté.

Remarque: les { } ne sont pas nécessaires lorsque le bloc ne comporte qu'une seule instruction.

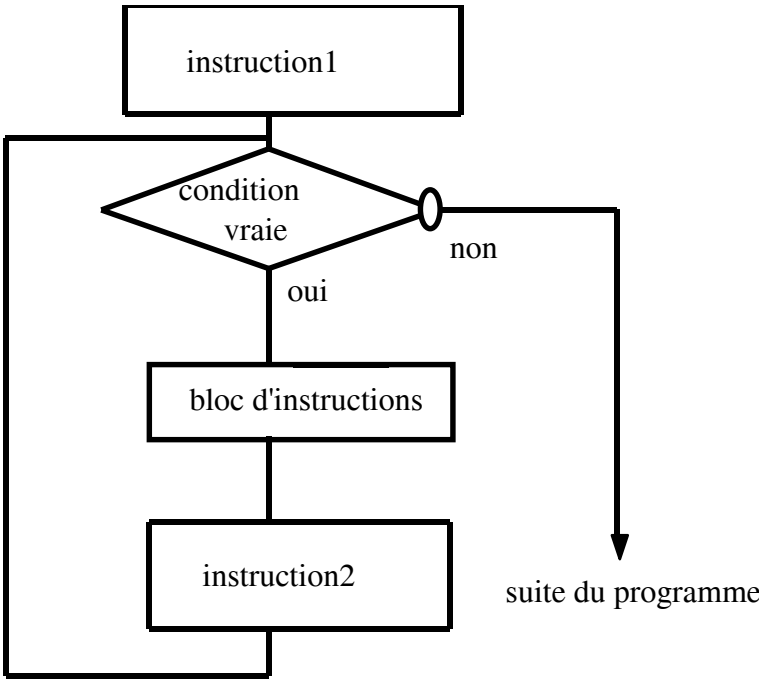
Remarque: On peut rencontrer la construction suivante: **while (expression);** terminée par un ; et sans la présence du bloc d'instructions. Cette construction signifie: **"tant que l'expression est vraie attendre". Ceci ne doit être exploité que par rapport à des événements externes au programme (attente de la frappe du clavier par exemple).**

L'INSTRUCTION POUR ...

Il s'agit de l'instruction:

pour (instruction1; condition; instruction2)
{BLOC D'INSTRUCTIONS}

Organigramme:



Syntaxe en C:

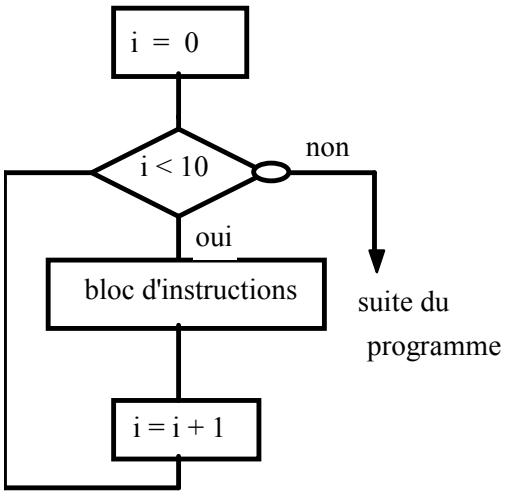
```
for(instruction1; condition; instruction2)
                    {
                    .....;          // bloc d'instructions
                    .....;
                    .....;
                    }
                    suite du programme ...
```

Remarques:
Il s'agit d'une version enrichie du « while ».
Les {} ne sont pas nécessaires lorsque le bloc ne comporte qu'une seule instruction.
Les 3 instructions du for ne portent pas forcément sur la même variable.
Une instruction peut être omise, mais pas les ;

Exemples:

```
for(int i = 0 ; i<10 ; i++)
{
.....;          // bloc d'instructions
.....;
.....;
}
suite du programme ...
```

correspond à l'organigramme suivant:



La boucle
```
for(;;)
{
.....;              // bloc d'instructions
.....;
.....;
}
```
est une boucle infinie (répétition infinie du bloc d'instructions).

Utilisation de variables différentes:

```
resultat = 0;
for(int i = 0 ; resultat<30 ; i++)
        {
        .....;              // bloc d'instructions
        .....;
        .....;
        resultat = resultat + 2*i;
        }
```

Exercice III-3:
Ecrire un programme permettant de saisir un entier n, de calculer n!, puis de l'afficher.
Utiliser une boucle do ...while puis while puis for.
Quelle est la plus grande valeur possible de n, si n est déclaré int, puis unsigned int?

Exercice III-4:
La formule récurrente ci-dessous permet de calculer la racine du nombre 2 :
$U_0 = 1$
$U_i = (U_{i-1} + 2/U_{i-1}) / 2$
Ecrire un programme qui saisit le nombre d'itérations, puis calcule et affiche la racine de 2.

L'INSTRUCTION AU CAS OU ... FAIRE ...

L'instruction switch permet des choix multiples **uniquement sur des entiers (int) ou des caractères (char).**

Syntaxe:

```
switch(variable de type char ou int)        au cas où la variable vaut:
{
case  valeur1: ......;                      - cette valeur1: exécuter ce bloc d'instructions.
               .......;
               break;                       - se brancher à la fin du bloc case.
      valeur2:.......;                      - cette valeur2:  exécuter ce bloc d'instructions.
               ........;
               break;                             - se brancher à la fin du bloc case.
       .
       .                                          etc ...
       .
    default:  .......;                      - aucune des valeurs précédentes: exécuter ce bloc
              ........;                     d'instructions, pas de "break" ici.

}
```

le bloc "default" n'est pas obligatoire.

L'instruction switch correspond à une cascade d'instructions if ...else

Exemple:
Cette instruction est commode pour fabriquer des "menus":

```
char choix;
cout<<"LISTE PAR GROUPE TAPER 1\n";
cout<<"LISTE ALPHABETIQUE TAPER 2\n";
cout<<"POUR SORTIR TAPER S\n";
cout<<"\nVOTRE CHOIX : ";
cin >> choix;
switch(choix)
{
case '1':  .......;
           .......;
          break;

case '2':  ......;
           ......;
           break;

case 'S': cout<<"\nFIN DU PROGRAMME ....";
           break;
```

```
default;  cout<<"\nCE CHOIX N'EST PAS PREVU ";  // pas de break ici
}
```

COMPLEMENT SUR LES TESTS

En langage C++, **une expression nulle de type entier (int) est fausse, une expression non nulle de type entier (int) est vraie.**

Exemples:

```
int a,b,c,delta;                    est équivalent à    int a,b,c,delta;
delta = b*b-4*a*c;                                      delta = b*b-4*a*c;
if(delta != 0)                                          if(delta)
{ ..... }                                               { ..... }
```

```
int a,b,c,delta;                    est équivalent à    int a,b,c,delta;
delta = b*b-4*a*c;                                      delta = b*b-4*a*c;
if(delta == 0)                                          if(!delta)
{ ..... }                                               {.....}
```

En langage C++, le type booléen a été introduit. Il prend les valeurs TRUE ou FALSE.
Par exemple :

```
bool test ;
test = (x<45) ;
if ( test == TRUE)
{……..}
```

```
ou plus simplement                  ou aussi
if( (x<45) == TRUE) )               if (x<45) // !!!!!
```

EXERCICES RECAPITULATIFS

Exercice III_5: résoudre $ax^2 + bx + c = 0$.

Exercice III_6: La fonction kbhit appartient à la bibiothèque conio.h. Une fonction équivalente peut exister avec d'autres compilateurs. La fonction kbhit teste si un caractère a été frappé au clavier. Tant que ce n'est pas vrai kbhit renvoie 0 (ceci signifie que la valeur retournée par la fonction kbhit est 0).

Exemple: **while(kbhit() == 0)** // tant qu'aucun caractère n'a été frappé exécuter la boucle

{ ..... }

Cette écriture est équivalente à:
while(!kbhit()) // tant que kbhit est faux, exécuter la boucle
{.....} Ecrire un programme qui affiche le carré des entiers 1, 2, 3 ....., toutes les 500 ms tant qu'aucun caractère n'a été frappé au clavier. Générer la temporisation à l'aide d'une boucle for et d'un décompteur.

CORRIGE DES EXERCICES

Exercice III-1:

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

    void main()
   {
      char c,d;
      cout <<"ENTRER UNE LETTRE MAJUSCULE :";
      cin>>c;

      d = c + 'a' - 'A';

      if((c>='A') && (c<='Z')) cout<<"LETTRE EN MINUSCULE : "<<d<<"\n";
      else cout<<"CE N'EST PAS UNE LETTRE MAJUSCULE\n";

      cout<<"POUR CONTINUER FRAPPER UNE TOUCHE ";
      getch();
   }
```

Exercice III-2:

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

    void main()
   {
      int I, V, M, P, Q;
      cout<<"Entrer le nombre d'inscrits : "; cin>>I;
      cout<<"Entrer le nombre de votants : "; cin>>V;
      cout<<"Entrer le quorum : "; cin>>Q;
      P = V*100/I;
      M = V/2 +1;
      if(P > Q)
      {
         cout<<"Quorum atteint - vote valable\n";
         cout<<"Participation: "<<P<<"% - Majorite obtenue pour : ";
         cout<< M <<" bulletins\n";
      }
      else
      {
         cout<<"Quorum non atteint - vote non valable\n";
      }
      cout<<"POUR CONTINUER FRAPPER UNE TOUCHE ";
      getch();
   }
```

Exercice III_3:

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

    void main()
   {
      int n, fac= 1;
      cout<<"ENTRER UN ENTIER : ";cin>>n;
      for (int i=1;i<=n;i++) fac= fac * i;
      cout<<"\nn ="<<n<<" n!= "<<fac;
      cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE ";
      getch();
   }
```

Les entiers sont des nombres de 32 bits:
n int : n! maximum= 12!
n unsigned int : n! maximum = 12!

Exercice III_4:

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

    void main()
   {
      int n,;
      float U0=1.0, Ui;
      cout <<"ENTRER LE NOMBRE D'ITERATIONS: ";cin >> n;
      for (int i=0;i<n;i++)
      {
         Ui = (U0 + 2.0/U0)/2.0;
         U0 = Ui;
      }

      cout <<"RESULTAT = "<< Ui <<"\n";
      cout <<"\nPOUR CONTINUER FRAPPER UNE TOUCHE ";

      getch();
   }
```

Exercice III_5:

```
#include <iostream.h>
#include <conio.h>
#include <math.h>// contient la fonction racine

    void main()
   {
      float a,b,c,x1,x2;
   // saisie de A, B et C
      cout<<"\t\t\tRESOLUTION DE L'EQUATION DU SECOND DEGRE\n";
      cout<<"\t\t\t                2\n";
      cout<<"\t\t\t             AX +BX+C=0\n\n\n";
      cout<<"SAISIR A B C SEPARES PAR RETURN\n";
      cout<<"A = "; cin>>a;
      cout<<"B = "; cin>>b;
      cout<<"C = "; cin>>c;

   // début du calcul : cas particuliers
      if((a==0)&&(b==0)&&(c==0))cout<<"INFINITE DE SOLUTIONS\n";
      else if((a==0)&&(b==0)&&(c!=0))cout<<"PAS DE SOLUTIONS\n";
      else if((a==0)&&(b!=0))cout<<"UNE SOLUTION: X= "<<-c/b<<"\n";
      else //cas général
      {_// delta : variable "brouillon"
         float delta = b*b-4*a*c;  // déclarée localement
         cout<<"DELTA="<<delta<<"\n";
         if(delta<0)cout<<"DELTA NEGATIF PAS DE SOLUTION\n";
         else
         {
            if(delta==0)cout<<"DELTA NUL, UNE SOLUTION X="<<-b/2/a<<"\n";
            else
            {
               x1= (-b+sqrt(delta))/2/a;
               x2= (-b-sqrt(delta))/2/a;
               cout<<"DELTA POSITIF DEUX SOLUTIONS\n";
               cout<<"X1= "<<x1<<"  X2= "<<x2<<"\n";
            }
         }
      }

   // calculs terminés
      cout<<"\n\nPOUR CONTINUER FRAPPER UNE TOUCHE";
      getch();
   }
```

Exercice III_6:

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

    void main()
   {
      int i = 0;
   // le type float pour générer des temporisations suffisamment longues
      float x, tempo=5000000.0;
      cout<<"POUR SORTIR DE CE PROGRAMME FRAPPER UNE TOUCHE ...\n";
      do
      {
         cout<<"i = "<< i <<"  i*i="<< i*i <<"\n";
         for(x=tempo; x>0; x--);
         i++;
      }
      while(kbhit()==0); // on peut aussi écrire while(!kbhit());
   }
```

COURS et TP DE LANGAGE C++

Chapitre 4

Utilisation d'une bibliothèque

Joëlle MAILLEFERT

joelle.maillefert@iut-cachan.u-psud.fr

IUT de CACHAN

Département GEII 2

CHAPITRE 4

UTILISATION D'UNE BIBLIOTHEQUE

Ce petit chapitre vise à expliquer comment se servir d'une bibliothèque de fonctions. On prendra quelques exemples dans la bibliothèque de BORLAND C++.

NOTION DE PROTOTYPE

Les fichiers de type ".h" (conio.h, dos.h iostream.h etc...), appelés ***fichiers d'en tête*** contiennent la définition des ***prototypes*** des fonctions utilisées dans le programme. Le prototype précise la syntaxe de la fonction: son nom, le type des paramètres éventuels à passer, le type de l'expression - la valeur retournée au programme (void si aucun paramètre retourné).
Grâce aux lignes "#include", le compilateur lit les fichiers de type ".h" et vérifie que la syntaxe de l'appel à la fonction est correcte.

FONCTIONS NE RENVOYANT RIEN AU PROGRAMME

Ce sont les fonctions de type **void.**

Exemple: **clrscr**

fonction Efface l'écran de la fenêtre dos.

prototype void clrscr();

prototype dans conio.h et donc bibliothèque à charger.

Une fonction ne renvoyant rien (de type void) s'écrit telle que. C'est une action représentée par une instruction. Ici pas de passage d'arguments.

```
ex:   clrscr();              // efface l'écran
      cout >>"BONJOUR\n";
      cout>>"AU REVOIR\n";
```

FONCTIONS RENVOYANT UNE VALEUR AU PROGRAMME

Ce sont les fonctions de type **autre que void.** Elles renvoient au programme une valeur (expression) dont le type est précisé dans la documentation.

Exemple: **kbhit**

fonction	Teste une éventuelle frappe au clavier
prototype	int kbhit();
prototype dans	conio.h et donc bibliothèque à charger.

La fonction kbhit renvoie un entier au programme appelant. Cet entier vaut 0, tant qu'il n'y a pas eu de frappe clavier. On ne peut donc pas écrire la fonction telle que. Il faut la traiter comme le résultat de calcul d'une expression qui peut être mémorisée dans une variable de type int.

ex:
```
int u;
do
{
.......
........
u = kbhit();
}
while(u== 0);
// boucle exécutée tant qu'il n'y a pas de frappe clavier
```

ou plus simplement

```
do
{
.......
........
}
while(kbhit()== 0);
```

FONCTIONS AVEC PASSAGE DE PARAMETRE

Exemple: **log**

fonction	Fonction logarithme népérien..
prototype	double log(double x);
prototype dans	math.h et donc bibliothèque à charger.

La fonction log, renvoie un réel au programme appelant. On traite la fonction comme une variable de type double. Il faut lui passer un paramètre effectif de type double. On peut se contenter d'un paramètre effectif de type float (en C et C++ le compilation fait automatiquement une transformation de type qui n'est pas dégradante dans ce cas précis).

ex:
```
double x, y;
cout <<"SAISIR x : ";
cin >> x;
```

y = log(x);
cout<<"log(x)= "<<y<<"\n";

Exercice IV_1:

En utilisant randomize et random jouer au 421.

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

    void main()
   {
      char c;
      int n1, n2, n3;
      cout<<"JEU DU 421\n";
      randomize();
      do{
         clrscr();
      // LANCEMENT DES DES
         cout<<"LANCER LES DES EN FRAPPANT UNE TOUCHE : ";
         getch();
         n1 = random(6) + 1;
         n2 = random(6) + 1;
         n3 = random(6) + 1;
         cout<<"\n VOICI LES DES : "<< n1 <<" "<< n2 <<" " << n3 <<"\n";
      // TEST
         if(((n1==4) && (n2==2) && (n3 ==1))||
         ((n1==4) && (n2==1) && (n3 ==2))||
         ((n1==2) && (n2==4) && (n3 ==1))||
         ((n1==2) && (n2==1) && (n3 ==4))||
         ((n1==1) && (n2==2) && (n3 ==4))||
         ((n1==1) && (n2==4) && (n3 ==2)))cout<<"GAGNE !\n";
         else cout <<"PERDU !\n";
         cout<<"\nPOUR REJOUER FRAPPER O SINON UNE TOUCHE QUECONQUE\n";
         c = getch();
      }
      while((c=='O')||(c=='o'));
   }
```

COURS et TP DE LANGAGE C++

Chapitre 5

Les pointeurs

Joëlle MAILLEFERT

joelle.maillefert@iut-cachan.u-psud.fr

IUT de CACHAN

Département GEII 2

CHAPITRE 5

LES POINTEURS

Que ce soit dans la conduite de process, ou bien dans la programmation orientée objet, l'usage des pointeurs est extrêmement fréquent en C++.

L'OPERATEUR ADRESSE &

L'opérateur adresse & indique l'adresse d'une variable en mémoire.

Exemple:
```
int i = 8;
cout<<"VOICI i:"<<i;
cout<<"\nVOICI SON ADRESSE EN HEXADECIMAL:"<<&i;
```

Exercice V_1: Exécuter l'exemple précédent, et indiquer les cases-mémoire occupées par la variable i.

LES POINTEURS

Définition: Un pointeur est une adresse. On dit que le pointeur pointe sur une variable dont le type est défini dans la déclaration du pointeur. Il contient l'adresse de la variable.

DECLARATION DES POINTEURS

Une variable de type pointeur se déclare à l'aide du type de l'objet pointé précédé du symbole ***.**

Exemple:
```
char *pc;    pc est un pointeur sur un objet de type char
int *pi;     pi est un pointeur sur un objet de type int
float *pr;   pr est un pointeur sur un objet de type float
```

Excercice V_1: Modifier l'exercice 1 comme ci-dessous, et conclure :

```
int i = 8;
int *p;    // déclaration d'un pointeur sur entier
cout<<"VOICI i : "<<i;
cout<<"\nVOICI SON ADRESSE EN HEXADECIMAL : "<<&i;

p = &i;   // p contient l'adresse de i, ont peut dire  qu'il pointe sur i
cout<<"\nVOICI SON ADRESSE EN HEXADECIMAL : "<<p;
```

MANIPULATION DE LA CASE MEMOIRE

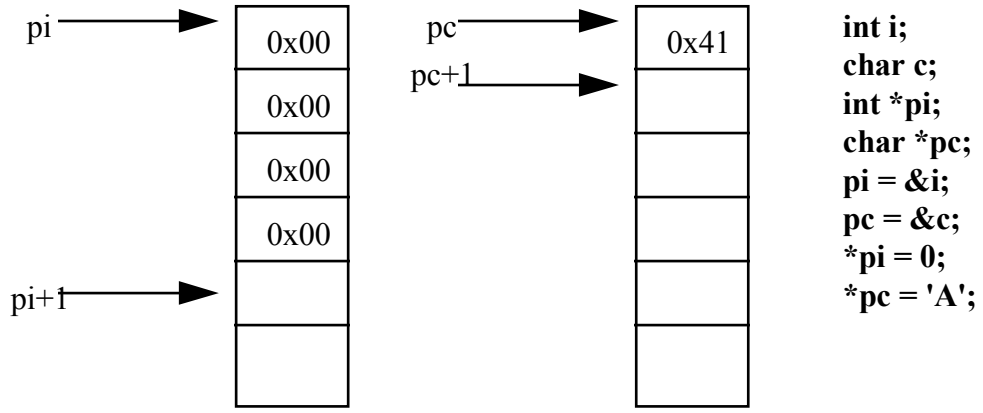
En dehors de la partie déclarative, l'opérateur * désigne en fait le contenu de la case mémoire pointée.

Excercice V_1: Modifier l'exercice 1 comme ci-dessous, et conclure :

```
int i = 8;
int *p;
cout<<"VOICI i :"<<i;
cout<<"\nVOICI SON ADRESSE EN HEXADECIMAL :"<<&i;

p = &i;
cout<<"\nVOICI SON ADRESSE EN HEXADECIMAL :"<<p;
cout<<"\nVOICI i :"<<*p;

*p = 23;
cout<<"\nVOICI i:"<<i;
cout<<"\nVOICI i:"<<*p;
```

ARITHMETIQUE DES POINTEURS

On peut essentiellement **déplacer** un pointeur dans un plan mémoire à l'aide des opérateurs d'addition, de soustraction, d'incrémentation, de décrémentation. On ne peut le déplacer que **d'un nombre de cases mémoire multiple du nombre de cases réservées en mémoire pour la variable sur laquelle il pointe.**



Exemples:

```
int *pi;        //  pi pointe sur un objet de type entier
float *pr;      // pr pointe sur un objet de type réel
char *pc;       //  pc pointe sur un objet de type caractère

*pi = 421;      //  421 est le contenu de la case mémoire p et des 3 suivantes
*(pi+1) = 53;  // on range 53,  4 cases mémoire plus loin
*(pi+2) = 0xabcd;     // on range 0xabcd  8 cases mémoire plus loin
*pr = 45.7;     // 45,7 est rangé dans la case mémoire r et les 3 suivantes
*pc = 'j';      // le contenu de la case mémoire c est le code ASCII de 'j'
```

AFFECTATION D'UNE VALEUR A UN POINTEUR ET ALLOCATION DYNAMIQUE

Lorsque l'on déclare une variable char, int, float .... un nombre de cases mémoire bien défini est **réservé** pour cette variable. En ce qui concerne les pointeurs, l'allocation de la case-mémoire pointée obéit à des règles particulières :

Exemple:
char *pc;
Si on se contente de cette déclaration, le pointeur pointe « n'importe où ». Son usage, tel que, dans un programme peut conduire à un « plantage » du programme ou du système d'exploitation si les mécanismes de protection ne sont pas assez robustes.
L'initialisation du pointeur n'ayant pas été faite, on risque d'utiliser des adresses non autorisées ou de modifier d'autres variables.

Exercice V_2 : Tester le programme ci-dessous et conclure

```
char *pc;       // normalement, il faudrait réserver !
*pc = 'a';      // le code ASCII de a est rangé dans la case mémoire pointée par pc
*(pc+1) = 'b'; // le code ASCII de b  est rangé une case mémoire plus loin
*(pc+2) = 'c'; // le code ASCII de c est rangé une case mémoire plus loin
*(pc+3) = 'd'; // le code ASCII de d est rangé une case mémoire plus loin
cout<<"Valeurs :"<<*pc<<" "<<*(pc+1)<<" "<<*(pc+2)<<" "<<*(pc+3);
```

1ère méthode : utilisation de l'opérateur adresse :

C'est la méthode qui a été utilisée dans l'exercice V_1. Dans ce cas, l'utilisation de pointeurs n'apporte pas grand-chose par rapport à l'utilisation classique de variables.

2ème méthode : affectation directe d'une adresse :

Cette méthode est réservée au contrôle de processus, quand on connaît effectivement la valeur de l'adresse physique d'un composant périphérique.
Elle ne fonctionne pas sur un PC, pour lequel les adresses physiques ne sont pas dans le même espace d'adressage que les adresses mémoires.

Il faut utiliser l'opérateur de "cast", jeu de deux parenthèses.
char *pc;
pc = (char*)0x1000; // p pointe sur l'adresse 0x1000

3ème méthode : allocation dynamique de mémoire :

C'est la méthode la plus utilisée et qui donne pleinement leur intérêt aux pointeurs par rapport aux variables classiques.

Via l'opérateur ***new***, natif dans le C++, on réserve, pendant l'exécution du programme, de la place dans la mémoire pour l'objet (ou la variable) pointé. L'adresse de base est choisie par le système lors de l'exécution, en fonction du système d'exploitation.
Le programmeur n'a à se soucier que de la quantité de cases mémoire dont il a besoin.

Exercice V_2 : Modifier l'exercice V_2 comme ci-dessous :

```
char *pc;
pc = new char[4];    // réservation de place dans la mémoire pour 4 char

*pc = 'a';       // le code ASCII de a est rangé dans la case mémoire pointée par pc
*(pc+1) = 'b'; // le code ASCII de b  est rangé une case mémoire  suivante
*(pc+2) = 'c'; // le code ASCII de c est rangé une case mémoire  suivante
*(pc+3) = 'd'; // le code ASCII de d est rangé une case mémoire  suivante

cout<<"Valeurs :"<<*pc<<" "<<*(pc+1)<<" "<<*(pc+2)<<" "<<*(pc+3);

delete pc; // libération de la place
```

Remarques :

- L'allocation dynamique peut se faire n'importe quand dans le programme (avant d'utiliser le pointeur !).
- L'opérateur ***delete*** libère la place réservée quand elle devient inutile.
- L'allocation dynamique devrait se faire dès la déclaration du pointeur avec la syntaxe suivante :

char *pc = new char[4];

- Si on a besoin d'une seule case mémoire, on écrira :

char *pc; ou bien **char *pc = new char;**
pc = new char ;

Réservation de place pour d'autres types de pointeurs :

```
char *pc;
int *pi,*pj;
float *pr;
pc = new char[10];   // on réserve de la place pour 10 caractères, soit 10 cases mémoires
pi = new int[5];     // on réserve de la place pour 5 entiers, soit 20 cases mémoire
pj = new int;        // on réserve de la place pour 1 entier, soit 4 cases mémoire
pr = new float[6];   // on réserve de la place pour 6 réels, soit 24 cases mémoire

delete pc;           // on libère la place précédemment réservée pour c
delete pi;           // on libère la place précédemment réservée pour i
delete pr;           // on libère la place précédemment réservée pour r
```

Comme précédemment, l'allocation dynamique peut être faite dès la déclaration du pointeur, elle est préférable :

```
int  *pi = new int[5];
```

Exercice V_3:

adr1 et adr2 sont des pointeurs pointant sur des réels. La variable pointée par adr1 vaut -45,78; la variable pointée par adr2 vaut 678,89. Ecrire un programme qui affiche les valeurs de adr1, adr2 et de leur contenu.

L'opérateur de "cast", permet d'autre part, à des pointeurs de types différents de pointer sur la même adresse.

Exemple:

```
char *pc;      // pc pointe sur un objet de type caractère
int *pi;       // pi pointe sur un objet de type entier
pi = new int;  // allocation dynamique pour i
pc = (char*)i;// c et i pointent sur la même adresse, c sur un caractère
```

Exercice V_4:

adr_i est un pointeur de type entier; la variable pointée i vaut 0x41424344. A l'aide d'une conversion de type de pointeur, écrire un programme montrant le rangement des 4 octets en mémoire.

Exercice V_5:

Saisir un texte de 10 caractères. Ranger les caractères en mémoire. Lire le contenu de la mémoire et compter le nombre de lettres e.

Exercice V_6:

Saisir 6 entiers et les ranger à partir de l'adresse adr_deb. Rechercher le maximum, l'afficher ainsi que sa position. La place nécessaire dans la mémoire peut-être le résultat d'un calcul :

int taille;
char *image;
cout<<"\nQuelle est la taille de l'image ? (en octets)";
cin>>taille;
image = new char[taille];

Exercice V_7:

Reprendre l'exercice V_6 mais en demandant à l'utilisateur combien de nombres il souhaite traiter, et en réservant en mémoire la place nécessaire.

CORRIGE DES EXERCICES

Exercice V_3:

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

    void main()
   {
      float *adr1 = new float,*adr2 = new float;

      *adr1 = -45.78;
      *adr2 = 678.89;

      cout<<"adr1 = "<<adr1<<" adr2 = "<<adr2;
      cout<<"  r1 = "<<*adr1<<"  r2 = "<<*adr2;

      delete adr1;
      delete adr2;

      cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE ";

      getch();
   }
```

Exercice V_4:

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

    void main()
   {
      char *adr_c;
      int *adr_i = new int;

      char u;

      *adr_i = 0x41424344;
      adr_c = (char*)adr_i;

      u = *adr_c;
      cout<<" CONTENU : "<< u <<"\n";

      u = *(adr_c+1);
      cout<<" CONTENU : "<< u <<"\n";

      u = *(adr_c+2);
      cout<<" CONTENU : "<< u <<"\n";

      u = *(adr_c+3);
      cout<<" CONTENU : "<< u <<"\n";

      cout<<"Frapper une touche" ;

      getch();
   }
```

L'analyse de l'exécution de ce programme, montre que les microprocesseurs INTEL rangent en mémoire d'abord les poids faibles d'une variable.

Exercice V_5:

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

    void main()
   {
      char *adr_deb,c;
      int i,imax, compt_e = 0;
      adr_deb = new char[10];// texte de 10 caractères

   // saisie et rangement du texte

      cout<<"\nEntrer 10 caractères séparés par return:\n";
      for (i=0;i<10;i++)
      {
         cout<<"caractères numéro "<<i<<":";
         cin>>c;
         *(adr_deb + i) = c;
      }

   // lecture de la mémoire et tri

      for (i=0;i<10;i++)
      {
         c = *(adr_deb+i);
         cout<<"\nCARACTERE:"<<c;
         if (c=='e') compt_e++;
      }

   // résultats

      cout<<"\nNOMBRE DE e: "<<compt_e;
      delete adr_deb;
      cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE ";

      getch();
   }
```

Exercice V_6:

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

    void main()
   {
      int *adr_deb, i, max,position;
      adr_deb=new int[6];

   // saisie des nombres

      cout<<"SAISIE DES NOMBRES :\n";
      for(i=0;i<6;i++)
      {
         cout<<"ENTRER UN NOMBRE : ";
         cin>>*(adr_deb+i);
      }

   // tri

      max = *adr_deb;
      position = 1;
      for(i=0;i<6;i++)
      {
         if(*(adr_deb+i)>max)
         {
            max = *(adr_deb+i);
            position =  i+1;
         }
      }

   // résultats

      cout<<"MAXIMUM : "<<max<<" IL EST EN "<<position<<"EME POSITION\n";

      delete adr_deb;

      cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE";

      getch();
   }
```

Exercice V_7:

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

    void main()
   {
      int *adr_deb, i, max, position, nombre;

      cout<<"Combien de nombres voulez-vous traiter ?";
      cin>>nombre;
      adr_deb=new int[nombre];

   // saisie des nombres

      cout<<"SAISIE DES NOMBRES :\n";
      for(i=0;i<nombre;i++)
      {
         cout<<"ENTRER UN NOMBRE: ";
         cin>>*(adr_deb+i);
      }

   // tri

      max = *adr_deb;
      position = 1;

      for(i=0;i<nombre;i++)
      {
         if(*(adr_deb+i)>max)
         {
            max = *(adr_deb+i);
            position =  i+1;
         }
      }

   // résultats

      cout<<"MAXIMUM : "<<max<<" IL EST EN "<<position<<"EME POSITION\n";

      delete adr_deb;

      cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE";
      getch();
   }
```

COURS et TP DE LANGAGE C++

Chapitre 6

Les tableaux et les chaînes de caractères

Joëlle MAILLEFERT

joelle.maillefert@iut-cachan.u-psud.fr

IUT de CACHAN

Département GEII 2

CHAPITRE 6

LES TABLEAUX ET LES CHAINES DE CARACTERES

LES TABLEAUX DE NOMBRES (INT ou FLOAT)

Les tableaux correspondent aux vecteurs et matrices en mathématiques. Un tableau est caractérisé par sa taille et par le type de ses éléments.

Les tableaux à une dimension:

Déclaration: **type nom[dim];** Exemples: **int compteur[10];**
float nombre[20];

Cette déclaration signifie **que le compilateur réserve dim places en mémoire pour ranger les éléments du tableau.**

Exemples:
int compteur[10]; le compilateur réserve des places en mémoire pour 10 entiers, soit 40 octets.
float nombre[20]; le compilateur réserve des places en mémoire pour 20 réels, soit 80 octets.

Remarque: dim est nécessairement une EXPRESSION CONSTANTE (expression qui peut contenir des valeurs ou des variables constantes – c.f. modificateur const). Ce ne peut être en aucun cas une combinaison des variables du programme[1].

Utilisation: Un élément du tableau est repéré par son indice. En langage C et C++ les tableaux commencent à l'indice 0. L'indice maximum est donc dim-1.

Exemples:

```
compteur[2] = 5;
nombre[i] = 6.789;
cout<<compteur[i];
cin>>nombre[i];
```

Il n'est pas nécessaire de définir tous les éléments d'un tableau. Toutefois, les valeurs non initialisées contiennent alors des valeurs quelconques.

Exercice VI_1: Saisir 10 réels, les ranger dans un tableau. Calculer et afficher leur moyenne et leur écart-type.

[1] Sauf s'il s'agit d'une allocation dynamique en exploitant l'opérateur « new » - c.f. Cours n° 5.

Les tableaux à plusieurs dimensions:

Tableaux à deux dimensions:

Déclaration: **type nom[dim1][dim2];** Exemples: **int compteur[4][5];**
float nombre[2][10];

Utilisation: Un élément du tableau est repéré par ses indices. En langage C++ les tableaux commencent aux indices 0. Les indices maxima sont donc dim1-1, dim2-1.

Exemples:

```
compteur[2][4] = 5;
nombre[i][j] = 6.789;
cout<<compteur[i][j];
cin>>nombre[i][j];
```

Il n'est pas nécessaire de définir tous les éléments d'un tableau. Les valeurs non initialisées contiennent alors des valeurs quelconques.

Exercice VI_2: Saisir une matrice d'entiers 2x2, calculer et afficher son déterminant.

Tableaux à plus de deux dimensions:

On procède de la même façon en ajoutant les éléments de dimensionnement ou les indices nécessaires.

INITIALISATION DES TABLEAUX

On peut initialiser les tableaux au moment de leur déclaration:

Exemples:

int liste[10] = {1,2,4,8,16,32,64,128,256,528};

float nombre[4] = {2.67,5.98,-8.0,0.09};

int x[2][3] = {{1,5,7},{8,4,3}}; // 2 lignes et 3 colonnes

TABLEAUX ET POINTEURS

En déclarant un tableau, on dispose d'un pointeur (adresse du premier élément du tableau). Le nom d'un tableau est un pointeur sur le premier élément.

Les tableaux à une dimension:

Les écritures suivantes sont équivalentes:

/* Allocation dynamique pendant l'exécution du programme */ **int *tableau = new int[10];**	/* Allocation automatique pendant la compilation du programme */ **int tableau[10];**	**déclaration**
***tableau**	**tableau[0]**	**le 1er élément**
***(tableau+i)**	**tableau[i]**	**un élément d'indice i**
tableau	**&tableau[0]**	**adresse du 1er élément**
(tableau + i)	**&(tableau[i])**	**adresse d'un élément i**

Il en va de même avec un tableau de réels (float).

Remarques:

- La déclaration d'un tableau entraîne automatiquement la réservation de places en mémoire. C'est aussi le cas si on utilise un pointeur et l'allocation dynamique en exploitant l'opérateur **new**.
- On ne peut pas libérer la place réservée en mémoire pour un tableau créé en allocation automatique (la réservation étant réalisée dans la phase de compilation – en dehors de l'exécution). Par contre, en exploitant l'allocation dynamique via un pointeur, la primitive **delete** libère la mémoire allouée.

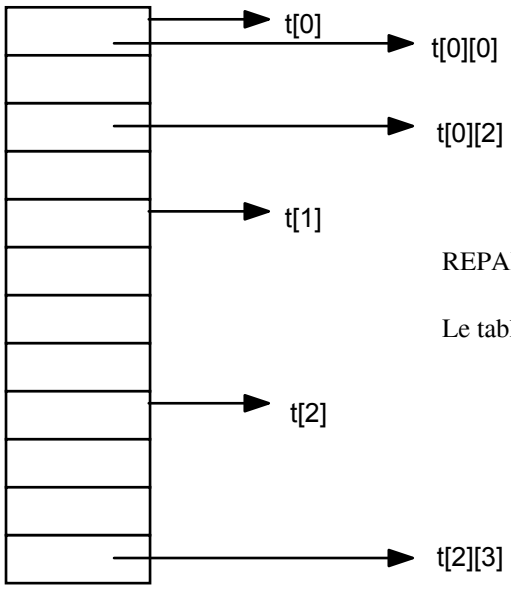
Les tableaux à plusieurs dimensions:

Un tableau à plusieurs dimensions est un pointeur de pointeur.

Exemple: **int t[3][4];** t est un pointeur de 3 tableaux de 4 éléments ou bien de 3 lignes à 4 éléments.

Les écritures suivantes sont équivalentes:

t[0]	**&t[0][0]**	**t**	**adresse du 1er élément**
t[1]	**&t[1][0]**		**adresse du 1er élément de la 2e ligne**
t[i]	**&t[i][0]**		**adresse du 1er élément de la ième ligne**
t[i]+1	**&(t[i][0])+1**		**adresse du 1er élément de la ième +1 ligne**



REPARTITION DES ELEMENTS EN MEMOIRE

Le tableau a été déclaréainsi: int t[3][4];

Exercice VI_3:

Un programme contient la déclaration suivante:
int tab[10] = {4,12,53,19,11,60,24,12,89,19};
Compléter ce programme de sorte d'afficher les adresses des éléments du tableau.

Exercice VI_4:

Un programme contient la déclaration suivante:
int tab[20] = {4,-2,-23,4,34,-67,8,9,-10,11, 4,12,-53,19,11,-60,24,12,89,19};
Compléter ce programme de sorte d'afficher les éléments du tableau avec la présentation suivante:

4	-2	-23	4	34
-67	8	9	-10	11
4	12	-53	19	11
-60	24	12	89	19

LES CHAINES DE CARACTERES

En langage C++, les chaînes de caractères sont des **tableaux de caractères**. Leur manipulation est donc analogue à celle d'un tableau à une dimension:

Déclaration: **char nom[dim];** ou bien **char *nom=new char[dim];**

Exemple: **char texte[20];** ou bien **char *texte=new char[20];**
Il faut toujours prévoir une place de plus pour une chaîne de caractères. En effet, en C ou en C++, une chaîne de caractère se termine toujours par le caractère **NUL ('\0')**. Ce caractère permet de détecter la fin de la chaîne lorsque l'on écrit des programmes de traitement.

Affichage à l'écran:
On utilise l'opérateur **cout** :
char texte[10] = « BONJOUR »;
cout<<"VOICI LE TEXTE:"<<texte;

Saisie:
On utilise l'opérateur **cin** :
char texte[10];
cout<<"ENTRER UN TEXTE: ";
cin>> texte;

Remarque: **cin** ne permet pas la saisie d'une chaîne comportant des espaces : les caractères saisis à partir de l'espace ne sont pas pris en compte (l'espace est un délimiteur au même titre que LF).
A l'issue de la saisie d'une chaîne de caractères, le compilateur ajoute '\0' en mémoire après le dernier caractère.

Exercice VI_5:

Saisir une chaîne de caractères, afficher les éléments de la chaîne caractère par caractère.

Exercice VI_6:

Saisir une chaîne de caractères. Afficher le nombre de lettres e de cette chaîne.

Fonctions permettant la manipulation des chaînes :

Les bibliothèques fournies avec les compilateurs contiennent de nombreuses fonctions de traitement des chaînes de caractères. En BORLAND C++, elles appartiennent aux bibliothèques **string.h ou stdlib.h.** En voici quelques exemples:

Générales (**string.h**):

Nom : **strcat**
Prototype : **void *strcat(char *chaine1, char *chaine2)**;
Fonctionnement : concatène les 2 chaînes, résultat dans chaine1, renvoie l'adresse de chaine1.
Exemple d'utilisation :

```
   char texte1[30] = "BONJOUR "; // remarquer l'espace après le R
   char texte2[20]= "LES AMIS";
   strcat(texte1,texte2);
// texte2 est inchangée, texte1 vaut maintenant "BONJOUR LES AMIS"
```

Nom : **strlen**
Prototype : **int strlen(char *chaine);**
Fonctionnement : envoie la longueur de la chaîne ('\0' non comptabilisé).
Exemple d'utilisation :

```
   char texte1[30] = "BONJOUR";
   int L = strlen(texte1);
   cout<< "longueur de la chaîne : "<<L; // L vaut 7
```

Nom: **strrev**
Prototype : **void *strrev(char *chaine);**
Fonctionnement : inverse la chaîne et, renvoie l'adresse de la chaîne inversée.
Exemple d'utilisation :

```
   char texte1[10] = "BONJOUR";
   strrev(texte1); // texte1 vaut maintenant "RUOJNOB"
```

Comparaison (**string.h**):

Nom : **strcmp**
Prototype : **int strcmp(char *chaine1, char *chaine2);**
Fonctionnement : renvoie un nombre:
- positif si chaine1 est supérieure à chaine2 (au sens de l'ordre alphabétique)
- négatif si chaîne1 est inférieure à chaine2
- nul si les chaînes sont identiques.

Cette fonction est utilisée pour classer des chaînes de caractères par ordre alphabétique.

Exemple d'utilisation :

```
char texte1[30] = "BONJOUR ";
char texte2[20]= "LES AMIS";
int n = strcmp(texte1,texte2); // n est positif
```

Copie (**string.h**):

Nom : strcpy
Prototype : **void *strcpy(char *chaine1,char *chaine2);**
Fonctionnement : recopie chaine2 dans chaine1 et renvoie l'adresse de chaîne1.
Exemple d'utilisation :

```
char texte2[20] = "BONJOUR ";
char texte1[20];
strcpy(texte1,texte2);
// texte2 est inchangée, texte1 vaut maintenant "BONJOUR "
```

Recopie (**string.h**):

Ces fonctions renvoient l'adresse de l'information recherchée en cas de succès, sinon le pointeur NULL (c'est à dire le pointeur dont la valeur n'a jamais été initialisée).

void *strchr(chaine, caractere); recherche le caractère dans la chaîne.
void *strrchr(chaine, caractere); idem en commençant par la fin.
void *strstr(chaine, sous-chaine); recherche la sous-chaîne dans la chaîne.

Exemple d'utilisation :

```
char texte1[30] ;
cout<< "SAISIR UN TEXTE :";
cin>>texte1;
if( strchr(texte1,A)!=NULL) cout<<"LA LETTRE A EXISTE DANS CE TEXTE ";
else cout<<<<  "LA LETTRE A NEXISTE PAS DANS CE TEXTE ";
```

Conversions (**stdlib.h**):

int atoi(char *chaine); convertit la chaîne en entier
float atof(char *chaine); convertit la chaîne en réel

Exemple d'utilisation :

```
  char texte[10]; int n;
  cout<<"ENTRER UN TEXTE: "; cin>>texte;
  n = atoi(texte) ;
  cout<<n;
// affiche 123 si texte vaut "123", affiche 0 si texte vaut "bonjour"
  void *itoa(int n, char *chaîne, int base);
// convertit un entier en chaîne:
// base: base dans laquelle est exprimé le nombre,
// cette fonction renvoie l'adresse de la chaîne.
```

Exemple d'utilisation :

```
  char texte[10];
  int n=12;
  itoa(12,texte,10); // texte vaut "12"
```

Pour tous ces exemples, la notation void signifie que la fonction renvoie un pointeur (l'adresse de l'information recherchée), mais que ce pointeur* ***n'est pas typé.*** *On peut ensuite le typer à l'aide de l'opérateur cast.*

Exemple:

```
  char *adr;
  char texte[10] = "BONJOUR";
  adr=(char*)strchr(texte,'N');//adr pointe sur l'adresse de la lettre N
```

Exercice VI_7:

L'utilisateur saisit le nom d'un fichier. Le programme vérifie que celui-ci possède l'extension .PAS

Exercice VI_8:

Un oscilloscope à mémoire programmable connecté à un PC renvoie l'information suivante sous forme d'une chaîne de caractères terminée par '\0'au PC:
"CHANNELA 0 10 20 30 40 30 20 10 0 -10 -20 -30 -40 -30 -20 -10 -0"
Afficher sur l'écran la valeur des points vus comme des entiers. On simulera la présence de l'oscilloscope en initialisant une chaîne de caractères char mesures [100].

CORRIGE DES EXERCICES

Exercice VI_1:

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

    void main()
   {
      float nombre[10], moyenne = 0, ecart_type = 0;
      int i;

   // saisie des nombres

      cout <<"SAISIR 10 NOMBRES SEPARES PAR RETURN :\n";
      for(i=0;i<10;i++)
      {
         cout<<"nombre["<<i<<"] = ";
         cin >>nombre[i];
      }

   // calcul de la moyenne

      for(i=0;i<10;i++)
      {
         moyenne = moyenne + nombre[i];
      }
      moyenne = moyenne/10;

   // calcul de l'écart type

      for(i=0;i<10;i++)
      {
         ecart_type =
            ecart_type + (nombre[i]-moyenne)*(nombre[i]-moyenne);
      }
      ecart_type = sqrt(ecart_type)/10; // racine

      cout<<"MOYENNE = "<<moyenne<<" ECART_TYPE = "<<ecart_type;

      cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE : ";
      getch();
   }
```

Exercice VI_2:

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

    void main()
   {
      int mat[2][2], det;

   // saisie

      cout<<"ENTRER SUCCESSIVEMENT LES VALEURS DEMANDEES : \n";
      cout<<"mat[0][0] = ";
      cin>>mat[0][0];
      cout<<"mat[1][0] = ";
      cin>>mat[1][0];
      cout<<"mat[0][1] = ";
      cin>>mat[0][1];
      cout<<"mat[1][1] = ";
      cin>>mat[1][1];

   // calcul

      det = mat[0][0]*mat[1][1]-mat[1][0]*mat[0][1];

   // affichage
      cout<<"DETERMINANT = "<<det;
      cout<<"\nPOUR CONTINUER FRAPPER  UNE TOUCHE: ";
      getch();
   }
```

Exercice VI_3:

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

    void main()
   {
      int tab[10]={4,12,53,19,11,60,24,12,89,19};
      cout<<"VOICI LES ELEMENTS DU TABLEAU ET LEURS ADRESSES:\n";
      for(int i=0;i<10;i++)
      {
         cout<<"ELEMENT NUMERO "<<i<<"="<<tab[i];
         cout<<" ADRESSE="<<tab+i<<"\n";
      }
      cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE: "; getch();
   }
```

Exercice VI_4:

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

    void main()
   {
      int tab[20] =
         {4,-2,-23,4,34,-67,8,9,-10,11, 4,12,-53,19,11,-60,24,12,89,19};
      cout<<"VOICI LE TABLEAU:\n\n";
      for(int i=0;i<20;i++)
         if (((i+1)%5)==0)
         {
            cout<<"\t"<<tab[i]<<"\n";
         }
         else
         {
            cout<<"\t"<<tab[i];
         }
      cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE : "; getch();
   }
```

Exercice VI_5:

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

    void main()
   {
      char i,*phrase = new char[20]; // réserve 20 places

      cout<<"ENTRER UNE PHRASE: ";
      cin>>phrase; // saisie

      cout<<"VOICI LES ELEMENTS DE LA PHRASE:\n";
      for(i=0;phrase[i] != '\0';i++)
      {
         cout<<"LETTRE : "<<phrase[i]<<"\n";
      }

      delete phrase;

      cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE : "; getch();
   }
```

Exercice VI_6:

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

    void main()
   {
      char *phrase = new char[20]; // réserve 20 places
      int compt_e = 0;

      cout<<"ENTRER UNE PHRASE:";
      cin>>phrase;  // saisie

      for(int i=0;phrase[i]!='\0';i++)
      {
         if(phrase[i]=='e')compt_e++;
      }
      cout<<"NOMBRE DE e : "<<compt_e;
      delete phrase;
      cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE "; getch();
   }
```

Exercice VI_7:

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

    void main()
   {
      char *nom = new char[30];
      char *copie = new char[30];
      int n;
      cout<<"\nNOM DU FICHIER (.PAS):"; cin>>nom;
      strcpy(copie,nom);
      strrev(copie); //chaîne inversée
      n = strnicmp("SAP.",copie,4);// n vaut 0 si égalité
      if(n!=0)cout<<"\nLE FICHIER N'EST PAS DE TYPE .PAS\n";
      else cout<<"\nBRAVO CA MARCHE\n";
      delete nom; delete copie;
      cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE "; getch();
   }
```

Exercice VI_8:

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

    void main()
   {
      char mesures[100] =
      "CHANNELA 0 10 20 30 40 30 20 10 0 -10 -20 -30 -40 -30 -20 -10 0";
      int i,j,val[20],nombre_val=0;
      char temp[4]; // chaîne temporaire

   // recherche des nombres

      for(i=9;mesures[i]!='\0';i++)
      {
         for(j=0;(mesures[i]!=' ')&&(mesures[i]!='\0');j++)
         {
            temp[j]=mesures[i];
            i++;
         }
         temp[j] = '\0'; // On borne la chaîne
         // Conversion de la chaîne temporaire en nombre
         val[nombre_val] = atoi(temp);
         nombre_val++;
      }

   // Affichage du résultat

      clrscr();
      for(i=0;i<nombre_val;i++)
      {
         cout<<"val["<<i<<"]="<<val[i]<<"\n";
      }

      cout<<"POUR SORTIR FRAPPER UNE TOUCHE : ";

      getch();
   }
```

COURS et TP DE LANGAGE C++

Chapitre 7

Les fonctions

Joëlle MAILLEFERT

joelle.maillefert@iut-cachan.u-psud.fr

IUT de CACHAN

Département GEII 2

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 réservé **return – retourner dans le programme appelant**).

Le programme principal (**void main()**), est une fonction parmi les autres. C'est le point d'entrée du programme obligatoire et unique.

Une variable connue uniquement d'une fonction ou est une variable locale.
Une variable connue de tous les programmes est une variable globale.

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

Elle est exécutée, mais le programme appelant ne reçoit aucune valeur de retour.

Exemple à expérimenter:

Exercice VII_1:

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

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

    void main()      // programme principal
   {
      // appel de la fonction, exécution à partir de sa 1ere ligne
      bonjour();
      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 exécutée.

*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() // déclaration de la fonction
   {
      cout<<"bonjour\n";
   }

    void coucou() // déclaration 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() // déclaration de la fonction
   {
      int n, n2; // variables locales à 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 carre. Elles sont appelées variables locales à carre. 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() // déclaration de la fonction
   {
      int n, n2;   // variables locales à carre
      cout<<"ENTRER UN NOMBRE : "; cin>>n;
      n2 = n*n;
      cout<<"VOICI SON CARRE : "<<n2<<"\n";
   }
```

```
 void cube()  // déclaration de la fonction
{
   int n, n3; // variables locales à 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 à 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 tous les programmes

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

    void cube()  // déclaration de la fonction
   {
      int n3;      // variable locale à 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 à 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 tous les programmes (fonctions et programme principal)
La variable locale choix n'est connue que du programme principal main.
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) doit posséder un minimum 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()  // déclaration de la fonction
   {  // random(n) retourne une valeur comprise entre 0 et n-1
      int test = random(6) + 1;  //variable locale
      return test;
   }

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

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 réservé *return.*

Exemple à expérimenter:

Exercice VII_8:

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

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

    void main()
   {
      int n1, n2, res1, res2; /* variables locales au main */
      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 formel, ou argument. Ce n'est pas une variable effective 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_9:

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_10:

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

    int somme(int *tx, int taille)
   {
      int total=0;
      for(int i=0;i<taille;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 = somme(tab, 20);
      cout<<"Somme des éléments du tableau : "<<resultat;
      cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE ";
      getch();
   }
```

Conclusion:

La tableau est passé au sous-programme comme un pointeur sur son premier élément.
On dit, dans ce cas, que l'on a passé le paramètre PAR ADRESSE. 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.

Exercice VII_11:

tab1 et tab2 sont des variables locales au programme principal.
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};
Ecrire une fonction de prototype **void affiche(int *tx, in taille)** qui permet d'afficher les nombres suivant un tableau par lignes de 5 éléments.
La mettre en oeuvre dans le programme principal pour afficher tab1 et tab2.

Exercice VII_12:

liste est une variable locale au programme principal
float liste[8] = {1.6,-6,9.67,5.90,345.0,-23.6,78,34.6};
Ecrire une fonction de prototype **float min(float *tx, int taille)** qui renvoie le minimum de la liste.
Ecrire une fonction de prototype **float max(float *tx, int taille)** qui renvoie le maximum de la liste.
Les mettre en oeuvre dans le programme principal.

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;    // initialisée à 0

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

    void main()
   {
      calcul(); cout<<"résultat= "<<resultat<<"\n";
      calcul(); cout<<"résultat= "<<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()
   {// la variable i n'est connue que du bloc for (norme définitive C++)
      for (int i= 0;i<10;i++ )
      {
         int j;// la variable j n'est connue que du bloc for
      }
   }
```

Cet exemple est équivalent, pour le résultat, à:

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

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 en utilisant un pointeur sur la variable (compatibilité avec le C_ANSI) ou par référence (spécifique au C++).

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 = 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'échange :\n";
      cout<<"a= "<<a<<"\n"; cout<<"b= "<<b<<"\n";

      ech(a,b);

      cout<<"\n\nValeurs de a et de b après l'échange:\n";
      cout<<"a= "<<a<<"\n"; cout<<"b= "<<b<<"\n";

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

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 puisque l'on fournit directement l'adresse du paramètre à la fonction. Il est impossible de passer les tableaux par référence. Le nom d'un tableau étant un pointeur son premier élément.

*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 chaîne 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;
// appel standard
   r1 = ma_fonction( i, r );
// appel correct, c est converti en int
   r2 = ma_fonction( c, r);
// appel correct, j est converti en float
   r3 = ma_fonction( i, j);
// appel correct, r est converti en int, j est converti en float
   r4 = ma_fonction( r, j);
}
```

Attention, ce type d'exploitation est possible mais dangereux.

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, initialisation par défaut 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); // les 2° et 3° 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 SURCHARGE DES FONCTIONS

Le C++ autorise la surcharge 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, ambigus, sont rejetés par le compilateur
      // test();
      // test (i,i);
      // test (r,r);
      // les initialisations 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 compilateur
      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 surcharge.

.

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

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

Exercice VII_9:

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

    int puissance(int x, int y)
   {
      int p=1;
      for(int 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_11:

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

    void affiche(int *tx ,int taille)
   {

      for(int i=0;i<taille;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(tab1,20);
      affiche(tab2,20);
      cout<<"\nPOUR CONTINUER FRAPPER UNE TOUCHE ";
      getch();
   }
```

Exercice VII_12:

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

    float max(float *tx,int taille)
   {
      float M = *tx;
      for(int i=1;i<taille;i++)
         if(*(tx+i)>M)
         {
            M = *(tx+i);
         }
      return(M);
   }

    float min(float *tx,int taille)
   {
      float m = *tx;
      for(int i=1;i<taille;i++)
         if(*(tx+i)<m)
         {
            m = *(tx+i);
         }
      return(m);
   }

    void main()
   {
      float liste[8] = {1.6,-6.0,9.67,5.90,345.0,-23.6,78.0,34.6};
      float resultat_min = min(liste,8);
      float resultat_max = max(liste,8);
      cout<<"LE MAXIMUM VAUT : "<<resultat_max<<"\n";
      cout<<"LE MINIMUM VAUT : "<<resultat_min<<"\n";
      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'éviter 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 contrôle d'erreur
   }
```

```
 void main()
{
   float r,c1,c2,c3;
   char *coul1 = new char[8];
   char *coul2 = new char[8];
   char *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<1000.0) cout<<"\nVALEUR DE R : "<<r<<" OHM\n";
      if((r>=1000.0)&&(r<999999.0))
      {
         r=r/1000;
         cout<<"\nVALEUR DE R: "<<(int)r<<" KOHM\n";
      }
      if(r>=999999.0)
      {
         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 référence 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-1];
         i++;
      }
      while(tx[i-1]!=13);
   }

    void affiche(int *tx)
   {
      cout<<"\n";
      for(int i=0; tx[i]!=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();
   }
```

COURS et TP DE LANGAGE C++

Chapitre 8

Les types de variables complexes

Joëlle MAILLEFERT

joelle.maillefert@iut-cachan.u-psud.fr

IUT de CACHAN

Département GEII 2

CHAPITRE 8

LES TYPES DE VARIABLES COMPLEXES

LES DECLARATIONS DE TYPE SYNONYMES : TYPEDEF

On a vu les types de variables utilisés par le langage C++: **char, int, float,** pointeur; le chapitre 9 traitera des fichiers (type **FILE**).

Le programmeur a la possibilité de créer ses propres types: **Il lui faut alors les déclarer** (après les déclarations des bibliothèques et les « define ») avec la syntaxe suivante:

Exemples:

typedef int entier; // on définit un nouveau type "entier" synonyme de "int"

typedef int vecteur[3]; // on définit un nouveau type "vecteur" synonyme
// de "tableau de 3 entiers"

typedef float *fpointeur; // on définit un nouveau type "fpointeur" synonyme */
// de "pointeur sur un réel

La portée de la déclaration de type dépend de l'endroit où elle est déclarée: dans main(), le type n'est connu que de main(); en début de programme, le type est reconnu dans tout le programme.

```
#include <iostream.h>

   typedef int   entier;
   typedef float point[2];

    void main()
   {
      entier n = 6;
      point xy;
      xy[0] = 8.6;
      xy[1] = -9.45;
   // etc ...
   }
```

Exercice VIII_1: Afficher la taille mémoire d'un point à l'aide de l'opérateur sizeof.

Exercice VIII_2: Définir le type **typedef char ligne[80] ;**

a- Déclarer dans le programme principal un pointeur de ligne; lui attribuer de la place en mémoire (pour 5 lignes). Saisir 5 lignes et les afficher.
b- Ecrire une fonction de prototype **void saisie (ligne *tx)** qui effectue la saisie de 5 lignes puis une autre fonction de prototype **void affiche (ligne *tx)** qui les affiche. Les mettre en oeuvre dans le programme principal

LES STRUCTURES

Les langages C et C ++ autorisent la déclaration de types particuliers: les structures. Une structure est constituée de plusieurs éléments de même type ou non (appelés **champs**). Le langage C++ a introduit une autre façon de définir plus simplement le type structure. Il est préférable de l'utiliser dans les programmes codés en C++.

Exemple:

Déclaration en C et C++	Déclaration spécifique à C++

`typedef // On définit un type struct`
`struct // identifié par fiche_t`
`{`
`char nom[10];`
`char prenom[10]; // Ici 4 champs`
`int age;`
`float note;`
`}`
`fiche_t;`
 	
/* en fin, on note _t pour bien signaler qu'il s'agit d'une définition de type et non d'un identificateur de variable */
	/* en fin, on note _t pour bien signaler qu'il s'agit d'une définition de type et non d'un identificateur de variable */
	
	`// On définit un type fiche_t`
	`struct fiche_t`
	`{`
	`char nom[10];`
	`char prenom[10];// Ici 4 champs`
	`int age;`
	`float note;`
	`};`

Utilisation:

On déclare des variables par exemple: **fiche_t f1, f2;**

puis, par exemple:
```
strcpy(f1.nom,"DUPONT");
strcpy(f1.prenom,"JEAN");
f1.age = 20;
f1.note = 11.5;
```

L'affectation globale est possible avec les structures: on peut écrire: **f2 = f1;**

Exercice VIII_3:

a- Déclarer la structure ci-dessus, saisir une fiche, afficher ses champs.
b- Même exercice mais en créant une fonction de prototype **void saisie(fiche_t &fx)** et une fonction de prototype **void affiche(fiche_t fx)**

STRUCTURES ET TABLEAUX

On peut définir un tableau de structures :

Exemple: (à partir de la structure définie précédemment)

Déclaration: **fiche_t f[10]; /* on déclare un tableau de 10 fiches */**

Utilisation: **strcpy(f[i].nom,"DUPONT") /* pour un indice i quelconque */**
strcpy(f[i].prenom,"JEAN");
f[i].age = 20;
f[i].note = 11.5;

Exercice VIII_4: Créer une structure **point{int num;float x;float y;}**
Saisir 4 points, les ranger dans un tableau puis les afficher.

STRUCTURES ET POINTEURS

On peut déclarer des pointeurs sur des structures. Cette syntaxe est très utilisée en langage C++, en raison des possibilités d'allocation dynamique de mémoire. **On adopte, pour la suite, la syntaxe de déclaration spécifique au C++.**
Un symbole spécial a été créé pour les pointeurs de structures, il s'agit du symbole **->**

Exemple: (à partir du type défini précédemment)

```
// Déclaration:
   fiche_t *f; // on déclare un pointeur sur fiche_t
// Utilisation:
   f = new fiche_t; // réserve de la mémoire pour une fiche
   strcpy(f->nom,"DUPONT");
   strcpy(f->prenom,"JEAN");
   f->age = 20;
   f->note = 11.5;
   delete f ;  // si on a terminé
```

Autre exemple :

```
// réserve de la mémoire pour 5 fiches
  fiche_t f = new fiche_t[5];

   for(int i=0 ;i<5 ;i++)
  {
     cin>> f[i]->nom ;
     cin>> f[i]->prenom ;
     cin>> f[i]->age ;
     cin>> f[i]->note ;
  }
  delete f ;  // si on a terminé
```

Exercice VIII_5:

a- Reprendre l'exercice VIII_4 en utilisant la notation « pointeur »
b- Même exercice mais en créant une fonction de prototype **void saisie(point_t *px)** et une fonction de prototype **void affiche(point_t *px)**

CORRIGE DES EXERCICES

Exercice VIII_1:

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

   typedef float point[2];

    void main()
   {
      cout<<"TAILLE D'UN POINT: "<<sizeof(point)<<"\n";
      cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE ";
      getch();
   }
```

Exercice VIII_2a:

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

   typedef char ligne[80];

    void main()
   {
      // réserve de la place pour 5 lignes
      ligne *texte = new ligne[sizeof(ligne)*5];

      cout<<"\n                SAISIE DU TEXTE\n\n";
      for (int i=0;i<5;i++)
      {
         cout<<"LIGNE Num "<<i<<"\n";
         cin>>texte[i]; // saisie de la ligne
      }

      cout<<"\n\n\n              AFFICHAGE DU TEXTE\n\n";
      for(i=0;i<5;i++)cout<<texte[i]<<"\n";
      delete texte;

      cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE ";
      getch();
   }
```

Exercice VIII_2b :

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

   typedef char ligne[80];

    void saisie (ligne *tx)
   {
      cout<<"\n                SAISIE DU TEXTE\n\n";
      for (int i=0;i<5;i++)
      {
         cout<<"LIGNE Num "<<i<<"\n";
         cin>>tx[i]; // saisie de la ligne
      }
   }

    void affiche(ligne *tx)
   {
      cout<<"\n\n\n              AFFICHAGE DU TEXTE\n\n";
      for(int i=0;i<5;i++)
         cout<<tx[i]<<"\n";
   }

    void main()
   {  // réserve de la mémoire pour 5 lignes
      ligne *texte = new ligne[sizeof(ligne)*5];
      saisie(texte);
      affiche(texte);
      delete texte;
      cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE ";
      getch();
   }
```

Exercice VIII_3a:

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

    struct fiche_t
   {
      char nom[10];
      char prenom[10];
      int age;
      float note;
   };
```

```
    void main()
   {
      fiche_t f;
      cout<<"SAISIE D'UNE FICHE \n";
      cout<<"NOM: ";     cin>>f.nom;
      cout<<"PRENOM: "; cin>>f.prenom;
      cout<<"AGE: ";     cin>>f.age;
      cout<<"NOTE: ";    cin>>f.note;
      cout<<"\n\nLECTURE DE LA FICHE :\n";
      cout<<"NOM: "<< f.nom <<" PRENOM : "<< f.prenom;
      cout<<" AGE: "<< f.age <<" NOTE: "<< f.note;
      cout<<"\n\nPOUR SORTIR FRAPPER UNE TOUCHE ";
      getch();
   }
```

Exercice VIII_3b:

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

    struct fiche_t
   {
      char nom[10];
      char prenom[10];
      int age;
      float note;
   };

    void saisie(fiche_t &fx)
   {  // passage par référence
      cout<<"SAISIE D'UNE FICHE \n";
      cout<<"NOM: ";     cin>>fx.nom;
      cout<<"PRENOM: "; cin>>fx.prenom;
      cout<<"AGE: ";     cin>>fx.age;
      cout<<"NOTE: ";    cin>>fx.note;
   }

    void affiche(fiche_t &fx)
   {  // passage par référence
      cout<<"\n\nLECTURE DE LA FICHE:\n";
      cout<<"NOM: "<< fx.nom <<" PRENOM : "<< fx.prenom;
      cout<<" AGE : "<< fx.age<<" NOTE: "<< fx.note;
   }
```

```
    void main()
   {
      fiche_t f;
      saisie(f);
      affiche(f);

      cout<<"\n\nPOUR SORTIR FRAPPER UNE TOUCHE ";
      getch();
   }
```

Exercice VIII_4:

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

    struct point_t
   {
      int num;
      float x;
      float y;
   };

    void main()
   {
      point_t p[4];// tableau de 4 points
   // saisie
      cout<<"SAISIE DES POINTS\n\n";
      for(int i=0;i<4;i++)
      {
         cout<<"\nRELEVE N° "<<i<<" :\n";
         p[i].num = i;
         cout<<"X= ";cin>>p[i].x;
         cout<<"Y= ";cin>>p[i].y;
      }
   // relecture
      cout<<"\n\nRELECTURE\n\n";
      for(i=0;i<4;i++)
      {
         cout<<"\nRELEVE Nø "<<p[i].num<<" :";
         cout<<"\nX= "<<p[i].x; cout<<"\nY= "<<p[i].y;
      }
      cout<<"\n\nPOUR SORTIR FRAPPER UNE TOUCHE ";
      getch();
   }
```

Exercice VIII_5a :

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

    struct point_t
   {
      int num;
      float x;
      float y;
   };

    void main()
   {
      point_t *p;          // pointeur sur point_t
      p = new point_t[4]; // réservation de mémoire

   // saisie

      cout<<"SAISIE DES POINTS\n\n";
      for(int i=0;i<4;i++)
      {
         cout<<"\nRELEVE Nø "<<i<<" :\n";
         p[i]->num = i;
         cout<<"X= ";cin>> p[i]->x;
         cout<<"Y= ";cin>> p[i]->y;
      }

   // relecture

      cout<<"\n\nRELECTURE\n\n";
      for(i=0;i<4;i++)
      {
         cout<<"\nRELEVE N° "<< p[i]->num<<" :";
         cout<<"\nX= "<< p[i]->x;
         cout<<"\nY= "<< p[i]->y;
      }

      delete p;    // libération de la mémoire

      cout<<"\n\nPOUR SORTIR FRAPPER UNE TOUCHE ";
      getch();
   }
```

Exercice VIII_5b :

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

    struct point_t
   {
      int num;
      float x;
      float y;
   };

// saisie
    void saisie(point_t *px)
   {
      cout<<"SAISIE DES POINTS\n\n";
      for(int i=0;i<4;i++)
      {
         cout<<"\nRELEVE Nø "<<i<<" :\n";
         px[i]->num = i;
         cout<<"X= ";cin>> px[i]->x;
         cout<<"Y= ";cin>> px[i]->y;
      }
   }

// relecture
    void affiche(point_t *px)
   {
      cout<<"\n\nRELECTURE\n\n";
      for(int i=0;i<4;i++)
      {
         cout<<"\nRELEVE N° "<< px[i]->num<<" :";
         cout<<"\nX= "<< px[i]->x;
         cout<<"\nY= "<< px[i]->y;
      }
   }

    void main()
   {  // pointeur sur tableau de 4 éléments de type point_t
      point_t *p = new point_t[4];

      saisie(p);
      affiche(p);

      delete p; // libération de la mémoire
      cout<<"\n\nPOUR SORTIR FRAPPER UNE TOUCHE "; getch();
   }
```

COURS et TP DE LANGAGE C++

Chapitre 9

Les fichiers

Joëlle MAILLEFERT

joelle.maillefert@iut-cachan.u-psud.fr

IUT de CACHAN

Département GEII 2

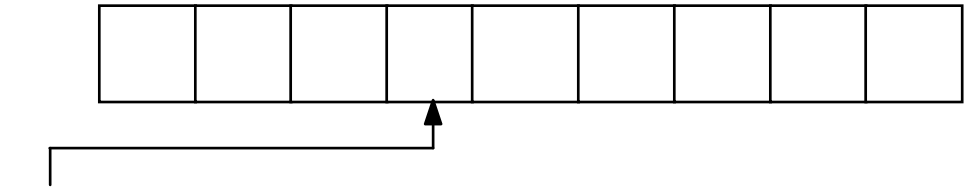
CHAPITRE 9

LES FICHIERS

GENERALITES

Un fichier est un ensemble d'informations stockées sur une mémoire de masse (disque dur, disquette, bande magnétique, CD-ROM).

Ces informations sont sauvegardées à la suite les unes des autres et ne sont pas forcément de même type (un char, un int, une structure ...)
Un **pointeur** permet de se repérer dans le fichier. On accède à une information en amenant le pointeur sur sa position.



pointeur

Sur le support de sauvegarde, le fichier possède un nom. Ce nom est composé de 2 parties : le nom proprement dit et l'extension. L'extension donne des informations sur le type d'informations stockées (à condition de respecter les extensions associées au type du fichier).

Exemples :

toto.txt	le fichier se nomme toto et contient du texte
mon_cv.doc	le fichier se nomme mon_cv et contient du texte, il a été édité sou WORD
ex1.cpp	le fichier se nomme ex1 et contient le texte d'un programme écrit en C++ (fichier source)
ex1.exe	le fichier se nomme ex1, il est exécutable
bibi.dll	le fichier se nomme bibi, c'est un fichier nécessaire à l'exécution d'un autre logiciel

Exercice IX_1 : Via l'explorateur de WINDOWS, reconnaître sur le disque dur du PC quelques fichiers.

On distingue généralement deux types d'accès:

1- Accès séquentiel (possible sur tout support, mais seul possible sur bande magnétique):

- Pas de cellule vide.
- On accède à une cellule quelconque en se déplaçant (via le pointeur sur un enregistrement du fichier), depuis la cellule de départ.
- On ne peut pas détruire une cellule.
- On peut par contre tronquer la fin du fichier.
- On peut ajouter une cellule à la fin.

2- Accès direct (RANDOM I/O) (Utilisé sur disques, disquettes, CD-ROM où l'accès séquentiel est possible aussi).

- Cellule vide possible.
- On peut directement accéder à une cellule.
- On peut modifier n'importe quelle cellule.

Il existe d'autre part deux façons de coder les informations stockées dans un fichier :

1- En binaire :

Fichier dit « binaire », les informations sont codées telles que. Ce sont en général des fichiers de nombres. Ils ne sont ni listables, ni éditables. Ils possèdent par exemple les extensions .OBJ, .BIN, .EXE, .DLL, .PIF etc …

Exercice IX_2 : Via le notepad ou l'éditeur de BC5, essayer d'éditer un fichier binaire.

2- en ASCII :

Fichier dit « texte », les informations sont codées en ASCII. Ces fichiers sont listables et éditables. Le dernier octet de ces fichiers est EOF (End Of File - caractère ASCII spécifique). Ils peuvent posséder les extensions .TXT, .DOC, .RTF, .CPP, .BAS, .PAS, .INI etc …

Exercice IX_3 : Via le notepad ou l'éditeur de BC5, essayer d'éditer quelques fichiers textes.

Un fichier possède des attributs, c'est à dire des droits d'accès : lecture, écriture (droit à modification), destruction etc…

Exercice IX_4 : Via le notepad, créer un fichier, y inscrire ce qui vous passe par la tête (1 ligne ou 2), le sauvegarder sous le nom **essai.dat** dans votre répertoire de travail, puis le fermer. Via l'explorateur de WINDOWS, et à l'aide du bouton droit de la souris, lire les attributs affectés par défaut par WINDOWS. Supprimer l'accès en écriture puis modifier le contenu du fichier et tenter une sauvegarde. Est-ce possible ?
Donner à nouveau l'accès en écriture et vérifier qu'une modification est possible.
Noter la taille du fichier fournie par WINDOWS et vérifier qu'elle correspond au nombre de caractères inscrits dans le fichier.

MANIPULATIONS GENERALES SUR LES FICHIERS

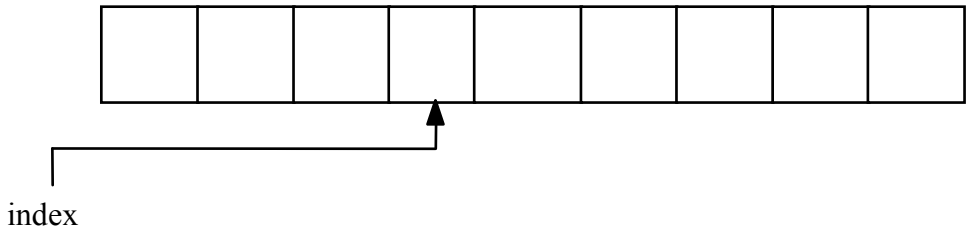
Opérations possibles avec les fichiers:
Créer - Ouvrir - Lire - Ecrire - Détruire – Renommer - Fermer.
La plupart des fonctions permettant la manipulation des fichiers sont rangées dans la bibliothèque standard STDIO.H, certaines dans la bibliothèque IO.H pour le BORLAND C++.
Ces fonctions sont très nombreuses. Seules quelques-unes sont présentées ici.

Pour manipuler un fichier, on commence toujours par l'ouvrir et vérifier qu'il est effectivement ouvert (s'il n'existe pas, cela correspond à une création).
Lorsque la manipulation est terminée, il faut fermer ce fichier et vérifier sa fermeture effective.

Le langage C++ ne distingue pas les fichiers à accès séquentiel des fichiers à accès direct, certaines fonctions de la bibliothèque livrée avec le compilateur permettent l'accès direct. Les fonctions standards sont des fonctions d'accès séquentiel.

1 - Déclaration: **FILE *index; // majuscules obligatoires pour FILE**

On définit un pointeur. Il s'agit du pointeur représenté sur la figure du début de chapitre. Ce pointeur repère une cellule donnée.



index est la variable qui permettra de manipuler le fichier dans le programme.

2 - Ouverture:

Il faut associer à la variable **index** au nom du fichier sur le support. On utilise la fonction **fopen** de prototype **FILE *fopen(char *nom, char *mode);**

On passe donc 2 chaînes de caractères
nom: celui figurant sur le support, par exemple: « a :\toto.dat »

mode (pour les fichiers TEXTES) :
« r » lecture seule
« w » écriture seule (destruction de l'ancienne version si elle existe)
« w+ » lecture/écriture (destruction ancienne version si elle existe)
« r+ » lecture/écriture d'un fichier existant (mise à jour), pas de création d'une nouvelle version.
« a+ » lecture/écriture d'un fichier existant (mise à jour), pas de création d'une nouvelle version, le pointeur est positionné à la fin du fichier.

mode (pour les fichiers BINAIRES) :
« rb » lecture seule
« wb » écriture seule (destruction de l'ancienne version si elle existe)
« wb+ »lecture/écriture (destruction ancienne version si elle existe)
« rb+ »lecture/écriture d'un fichier existant (mise à jour), pas de création d'une nouvelle version.
« ab+ »lecture/écriture d'un fichier existant (mise à jour), pas de création d'une nouvelle version, le pointeur est positionné à la fin du fichier.

A l'ouverture, le pointeur est positionné au début du fichier (sauf « a+ » et « ab+ », à la fin).

Exemple1 : **FILE *index ;**
index = fopen("a :\\toto.dat", "rb") ;

Exemple2 : **FILE *index ;**
char nom[30] ;
cout<< "Nom du fichier : " ;
cin >> nom ;
index = fopen(nom, "w") ;

3 - Fermeture:

On utilise la fonction de prototype **int fclose(FILE *f);**
Cette fonction retourne 0 si la fermeture s'est bien passée.

Exemple : **FILE *index ;**
index = fopen("a :\\toto.dat", "rb") ;
//
// Ici instructions de traitement
//
fclose(index) ;

Exercice IX_5 :

Ecrire un programme qui crée un fichier texte ("w") de nom **ex1.txt** dans le répertoire de travail, et qui le ferme.
Tester le programme puis vérifier la présence du fichier sur le disque dur. Quelle est sa taille ? Noter l'heure de création du fichier.

Exécuter à nouveau le programme et noter l'heure de création du fichier. A-t-elle changé ?

Remplacer l'attribut « w » par « r », exécuter le programme. L'heure de création du fichier a-t-elle changé ?

Sous WINDOWS, détruire le fichier.

Faire volontairement une faute de frappe dans le chemin d'accès au fichier, et exécuter le programme. Ceci provoque-t-il une erreur ?

Lorsqu'il y a un problème à l'ouverture du fichier, la fonction **fopen** retourne une valeur particulière du pointeur de fichier, la valeur NULL (ceci est une constante définie dans le fichier stdio.h). En testant la valeur retournée on peut ainsi réaliser un contrôle d'erreur :

Exemple : **FILE *index ;**
index = fopen("a :\\toto.dat", "rb") ;

if (index == NULL) cout << « Erreur a l'ouverture » ;
else
{
//
// Ici instructions de traitement

```
  //
fclose(index) ;
}
```

Exercice IX_6 : Modifier le programme de l'exercice IX_5 de sorte de vérifier si le chemin d'accès au fichier est correct.

4 - Destruction:

On utilise la fonction de prototype **int remove(char *nom);**
Cette fonction retourne 0 si la fermeture s'est bien passée.

Exemple1 : **remove(« a :\\toto.dat ») ;**

Exemple2 :
```
int x ;
x = remove("a :\\toto.dat") ;
if (x == 0) cout << « Fermeture OK : » ;
else cout << « Problème à la fermeture : » ;
```

5 - Renommer:

On utilise la fonction de prototype **int rename(char *oldname, char *newname);**
Cette fonction retourne 0 si la fermeture s'est bien passée.

Exemple1 : **rename("a :\\toto.dat" , "a :\\tutu.dat") ;**

Exemple2 :
```
int x ;
x = rename("a :\\toto.dat", "a :\\tutu.dat") ;
if (x == 0) cout << « Operation OK : » ;
else cout << "L'operation s'est mal passee : " ;
```

Exercice IX_7 : Modifier le programme de l'exercice IX_6 de sorte d'utiliser ces 2 dernières fonctions. Vérifier via l'explorateur de WINDOWS.

6 - Positionnement du pointeur au début du fichier:

On utilise la fonction de prototype **void rewind(FILE *f);**

Exemple :
```
FILE *index ;
index = fopen("a :\\toto.dat", "rb") ;  // pointeur au début

// ici traitement du fichier

rewind(index) ;   // repositionne le pointeur au début
```

7 - Fonction particulière aux fichiers à acces direct:

La fonction de prototype **int fseek(FILE *index , int offset , int direction)** déplace le pointeur de offset octets à partir de direction.
Valeurs possibles pour direction:

0 -> à partir du début du fichier.
1 -> à partir de la position courante du pointeur.
2 -> en arrière, à partir de la fin du fichier.

Cette fonction retourne « offset » si la manipulation s'est bien passée , retourne 0 si le pointeur n'a pu être déplacé.

Exemple : **FILE *index ;**
index = fopen("a :\\toto.dat", "rb") ; // pointeur au début

// ici manipulation du fichier

fseek(index, 5, 1) ; // déplace le pointeur de 5 position à partir de la
// position courante du pointeur

MANIPULATIONS DES FICHIERS TEXTES

1- Ecriture dans le fichier:

La fonction de prototype **int putc(char c, FILE *index)** écrit la valeur de c à la position courante du pointeur, le pointeur avance d'une case mémoire.
Cette fonction retourne –1 en cas d'erreur.

Exemple : **putc('A', index) ;**

La fonction de prototype **int fputs(char *chaîne, FILE *index)** est analogue avec une chaîne de caractères. Le pointeur avance de la longueur de la chaîne ('\0' n'est pas rangé dans le fichier).
Cette fonction retourne le code ASCII du caractère, retourne -1 en cas d'erreur (par exemple tentative d'écriture dans un fichier ouvert en lecture)

Exemple : **fputs("BONJOUR ! ", index) ;**

Exercice IX_8 : Modifier le programme de l'exercice IX_5 : Ouvrir le fichier, saisir quelques caractères, les ranger dans le fichier, puis saisir une chaîne de caractères et la ranger dans le fichier. Ne pas faire de contrôle d'erreur Vérifier via le notepad.

Ouvrir maintenant le fichier en mode "a". Exécuter le programme. Vérifier via le notepad que le fichier a bien été modifié.

Exercice IX_9 : Ouvrir maintenant le fichier en mode « r » et exploiter le contrôle d'erreur.

La fonction de prototype **int putw(int n, FILE *index)** écrit la valeur de n (codée en ASCII) à la position courante du pointeur, le pointeur avance d'une case mémoire.
Cette fonction retourne –1 en cas d'erreur.

Exemple : **int n = 45 ;**
putw(n, index) ;

2- Relecture d'un fichier:

Les fichiers texte se terminent par le caractère ASCII EOF (de code –1). Pour relire un fichier, on peut donc exploiter une boucle jusqu'à ce que la fin du fichier soit atteinte.

La fonction de prototype **int getc(FILE *index)** lit 1 caractère, et retourne son code ASCII, sous forme d'un entier. Cet entier vaut –1 (EOF) en cas d'erreur ou bien si la fin du fichier est atteinte. Via une conversion automatique de type, on obtient le caractère.

Exercice IX_10 (à expérimenter) :

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

    void main()
   {
      FILE *index;
      char c=0   ; // initialisation pour le 1er tour

      index = fopen("c:\\bc5\\sources\\ex1.txt","r");
      while(c!=EOF)
      {
         c=getc(index); // on utilise une conversion de type automatique
         cout<<c;
      }
      fclose(index);

      cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE ";
      getch();
   }
```

Exercice IX_11 : Copier un fichier dans un autre, vérifier via le notepad.

Exercice IX_12: Calculer et afficher le nombre de caractères d'un fichier texte.
Ecrire ensuite une fonction de prototype **int taille(char *nom)** qui retourne la taille de ce fichier et la mettre en œuvre dans le programme principal. Le fichier doit être ouvert et fermé dans la fonction. Le paramètre **nom** désigne le nom du fichier sur le disque dur. Il doit être fourni dans le programme principal et passé en paramètre.

La fonction de prototype **char *fgets(char *chaine, int n, FILE *index)** lit n-1 caractères à partir de la position du pointeur et les range dans chaîne en ajoutant '\0'. Retourne un pointeur sur la chaîne, retourne le pointeur NULL en cas d'erreur, ou bien si la fin du fichier est atteinte.

Exemple :
```
FILE *index ;
char texte[10] ;
// ouverture
fgets(texte, 7, index) ;  // lit 7 caractères dans le fichier et forme la chaîne
                          // « texte » avec ces caractères
```
La fonction de prototype **int getw(FILE *index)** lit 1 nombre stocké sous forme ASCII dans le fichier, et le retourne. Cet entier vaut –1 en cas d'erreur ou bien si la fin du fichier est atteinte.

MANIPULATIONS DES FICHIERS BINAIRES

La fonction **int feof(FILE *index)** retourne 0 tant que la fin du fichier n'est pas atteinte.

La fonction **int ferror(FILE *index)** retourne 1 si une erreur est apparue lors d'une manipulation de fichier, 0 dans le cas contraire.

La fonction de prototype **int fwrite(void *p, int taille_bloc, int nb_bloc, FILE *index)** écrit à partir de la position courante du pointeur **index** nb_bloc X taille_bloc octets lus à partir de l'adresse p. Le pointeur fichier avance d'autant.
Le pointeur p est vu comme une adresse, son type est sans importance.
Cette fonction retourne le nombre de blocs écrits (0 en cas d'erreur, ou bien si la fin du fichier est atteinte).

Exemple: taille_bloc = 4 (taille d'un entier en C++), nb_bloc=3, écriture de 3 entiers.
```
int tab[10] ;
fwrite(tab,4,3,index) ;
```

La fonction de prototype **int fread(void *p,int taille_bloc,int nb_bloc,FILE *index)** est analogue à **fwrite** en lecture.
Cette fonction retourne le nombre de blocs luts (0 en cas d'erreur, ou bien si la fin du fichier est atteinte).

Exercice IX_13: Créer et relire un fichier binaire de 3 entiers.
Ecrire ensuite une fonction de prototype **void creer(char *nom)** qui crée le fichier de 10 entiers et la mettre en œuvre dans le programme principal. Le fichier doit être ouvert et fermé dans la fonction. Le paramètre **nom** désigne le nom du fichier sur le disque dur. Il doit être fourni dans le programme principal et passé en paramètre.

Ecrire de même une fonction de prototype **void lire(char *nom)** qui relie le fichier et affiche son contenu. La mettre en œuvre dans le programme principal. Le paramètre **nom** désigne le nom du fichier sur le disque dur. Il doit être fourni dans le programme principal et passé en paramètre.

Ecrire maintenant une fonction de prototype **void ajout(char *nom, int n)** qui ajoute l'entier n au fichier précédent. La mettre en œuvre dans le programme principal. Le paramètre **nom** désigne le nom du fichier sur le disque dur. Il doit être fourni dans le programme principal et passé en paramètre. Relire ce fichier grâce à la fonction **void lire(char *nom).**

Ecrire maintenant une fonction de prototype **int cherche(char *nom, int n)** qui recherche si l'entier n existe dans le fichier précédent et relire le fichier grâce à la fonction **void lire(char *nom).** Cette fonction retourne la position du nombre si il existe, 0 sinon. La mettre en œuvre dans le programme principal. Le paramètre **nom** désigne le nom du fichier sur le disque dur. Il doit être fourni dans le programme principal et passé en paramètre.

Exercice IX_14: Créer une structure nom, prénom, âge. Ecrire un programme de gestion de fichier (binaire) avec menu d'accueil: possibilité de créer le fichier, de le lire, d'y ajouter une fiche, d'en rechercher une.

CORRIGE DES EXERCICES

Exercice IX_1:

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

    void main()
   {
      FILE *index;
      index = fopen("c:\\bc5\\sources\\ex1.txt","w");
      fclose(index);
      cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE "; getch();
   }
```

Exercice IX_2 :

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

    void main()
   {
      FILE *index;
      index = fopen("c:\\bc5\\sources\\ex1.txt","w");
      if(index == NULL) cout << "Erreur dans le chemin d'accès\n";
      else
      {
         cout << "Création du fichier OK\n";
         fclose(index);
      }
      cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE "; getch();
   }
```

Exercice IX_5 :

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

    void main()
   {
      FILE *index = fopen("c:\\bc5\\sources\\ex1.txt","w");
      fclose(index);
      cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE "; getch();
   }
```

Exercice IX_6 :

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

    void main()
   {
      FILE *index = fopen("c:\\bc5\\sources\\ex1.txt","w");
      if(index == NULL) cout << "Erreur dans le chemin d'accès\n";
      else
      {
         cout << "Création du fichier OK\n";
         fclose(index);
      }
      cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE ";
      getch();
      fclose(index);
   }
```

Exercice IX_7 :

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

    void main()
   {
      FILE *index = fopen("c:\\bc5\\sources\\ex1.txt","w");
      if(index == NULL) cout << "Erreur dans le chemin d'accès\n";
      else
      {
         cout << "Création du fichier OK\n";
         fclose(index);
      }
      rename( "c:\\bc5\\sources\\ex1.txt", "c:\\bc5\\sources\\change.txt");
      remove( "c:\\bc5\\sources\\ex1.txt");
      cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE ";
      getch();
      fclose(index);
   }
```

Exercice IX_8 :

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

    void main()
   {
      FILE *index = fopen("c:\\bc5\\sources\\ex1.txt","w");
      char c, texte[20];
      for(int i=0; i<5; i++)
      {
         cout<<"Saisir un caractere :"; cin>> c;
         putc(c, index);
      }
      cout<<"Votre message :"; cin>>texte;
      fputs(texte,index);
      fclose(index);
      cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE ";
      getch();
   }
```

Exercice IX_9 :

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

    void main()
   {
      FILE *index = fopen("c:\\bc5\\sources\\ex1.txt","r");
      char c   ;
      int  n;
      index = fopen("c:\\bc5\\sources\\ex1.txt","r");
      for(int i=0;i<3;i++)
      {
         cout<<"Saisir un caractere :"; cin>> c;
         n=putc(c, index);
         cout<<"n="<<n<<"\n";
      }
      fclose(index);
      cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE ";
      getch();
   }
```

Exercice IX_11 :

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

    void main()
   {
      FILE *index1, *index2;
      char c=0; // initialisation pour le 1er tour
      index1 = fopen("c:\\bc5\\sources\\ex1.txt","r");
      index2 = fopen("c:\\bc5\\sources\\ex2.txt","w");

      while(c!=EOF)
      {
         c=getc(index1);
         putc(c, index2);
      }
      fclose(index1);
      fclose(index2);

      cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE "; getch();
   }
```

Exercice IX_12 :

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

    int taille(char *nom)
   {
      FILE *index1 = fopen(nom,"r");
      int n=0;
      char c=0; // initialisation pour le 1er tour
      while(c!=EOF){
         c=getc(index1);
         n++;
      }
      fclose(index1);
      return n;
   }

    void main()
   {
      char mon_fichier[50]= "c:\\bc5\\sources\\ex1.txt";
      int resultat= taille(mon_fichier);
      cout<<"Taille du fichier :"<<resultat<<"\n";
      cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE "; getch();
   }
```

Exercice IX_13 :

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

    void creer(char *nom)
   {
      FILE *index = fopen(nom,"wb"); int n;
      for(int i=0;i<3;i++)
      {
         cout<<"Saisir un nombre: "; cin >> n ;
         fwrite(&n,4,1,index);
      }
      fclose(index);
   }

    void lire (char *nom)
   {
      FILE *index = fopen(nom,"rb"); int n, u=0  ;
      cout<<"Voici le contenu du fichier:\n";
      while(u==0)
      {
         fread(&n,4,1,index); u = feof(index);
         if(u==0)cout<< n <<" ";
      }
      cout<<"\n\n"; fclose(index);
   }

    void ajout (char *nom, int n)
   {
      FILE *index = fopen(nom,"ab+");
      fwrite(&n,4,1,index); fclose(index);
   }

    int cherche(char *nom, int n)
   {
      FILE *index = fopen(nom,"rb"); int place=0, u=0, n_lu, trouve=0;
      while(u==0)
      {
         fread(&n_lu,4,1,index); place++;
         if(n== n_lu) trouve = place;  // trouve le dernier
         u=feof(index);
      }
      fclose(index);
      return trouve;
   }

    void main()
   {
      char mon_fichier[50] = "c:\\bc5\\sources\\essai.dat";
      int plus, combien, numero;
      creer(mon_fichier); lire(mon_fichier);
      cout<<"Saisir le nombre à ajouter : "; cin>>plus;
      ajout(mon_fichier,plus); lire(mon_fichier);
      cout<<"Saisir le nombre a rechercher : "; cin>>combien;
      numero = cherche(mon_fichier, combien);
      if(numero==0)cout<<"Ce nombre n'existe pas\n";
      else cout<<"Ce nombre se trouve a la place numero "<<numero<<"\n";
      cout<<"\nPOUR SORTIR FRAPPER UNE TOUCHE "; getch();
   }
```

Exercice IX_14 :

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

   typedef
       struct
      {
         char nom[10];
         char prenom[10];
         int age;
      } carte;// création d'un type carte

    void creer_fichier(char *nom)
   {
      FILE *index = fopen(nom,"wb"); char choix; carte fiche;
      clrscr(); cout<<"CREATION DU FICHIER \n\n";
      do
      {
         cout<<"\nSAISIE D'UNE FICHE ?(o/n) "; choix = getch();
         if ((choix=='o')||(choix=='O'))
         {
            cout<<"\nNOM: ";  cin>>fiche.nom;
            cout<<"PRENOM: "; cin>>fiche.prenom;
            cout<<"AGE: ";    cin>>fiche.age;
            fwrite(&fiche,sizeof(carte),1,index);
         }
      }
      while((choix=='o')||(choix=='O'));
      fclose(index);
   }

    void lire_fichier(char *nom)
   {
      FILE * index = fopen(nom,"rb"); carte fiche; int compteur=0;
      clrscr(); cout<<"LECTURE DU FICHIER\n\n";
      if (index == NULL) cout<<"\nERREUR, CE FICHIER N'EXISTE PAS\n\n";
      else
      {
         cout<<"\nLISTING DU FICHIER\n\n";
         while(fread(&fiche,sizeof(carte),1,index)!=0)
         {
            cout<<"fiche numero "<<compteur<<" : "; compteur++;
            cout<<fiche.nom<<" "<<fiche.prenom<<" "<<fiche.age<<"\n";
         }
         fclose(index);
      }
      cout<<"POUR CONTINUER FRAPPER UNE TOUCHE "; getch();
   }
```

Suite page suivante :

```
 void recherche(char *nom)
{
   FILE *index = fopen(nom,"rb");
   carte fiche;
   int compteur=0;
   char trouve = 0, nn[10], pp[10];

   clrscr();
   cout<<"RECHERCHE DE FICHE\n\n";

   cout<<"\nFICHE A RETROUVER:\n";
   cout<<"NOM: ";   cin>>nn;
   cout<<"PRENOM: ";cin>>pp;

   while((fread(&fiche,sizeof(carte),1,index)!=0)&&(trouve==0))
   {
      if((strcmp(fiche.nom,nn)==0)&&(strcmp(fiche.prenom,pp)==0))
      {
         trouve=1;
         cout<<"FICHE RETROUVEE, NUMERO:"<<compteur<<"\n";
      }
      compteur++;
   }
   if (trouve==0)cout<<"CETTE FICHE N'EXISTE PAS\n";
   fclose(index);
   cout<<"POUR CONTINUER FRAPPER UNE TOUCHE ";
   getch();
}
```

Suite page suivante :

```
 void main()
{
   FILE *fichier;
   char mon_fichier[50]= "c:\\bc5\\sources\\ex14.dat";
   char choix;
   do
   {
      clrscr();
      cout<<"\t\t\tGESTION DE FICHIER\n";
      cout<<"\t\t\t------------------\n\n\n";
      cout<<"CREATION DU FICHIER  ---> 1\n";
      cout<<"LECTURE DU FICHIER   ---> 2\n";
      cout<<"AJOUTER UNE FICHE    ---> 3\n";
      cout<<"RECHERCHER UNE FICHE ---> 4\n";
      cout<<"SORTIE     ---> S\n\n";
      cout<<"VOTRE CHOIX: ";
      choix = getch();
      switch(choix)
      {
         case '1':creer_fichier(mon_fichier);
            break;
         case '2':lire_fichier(mon_fichier);
            break;
         case '3':ajout(mon_fichier);
            break;
         case '4':recherche(mon_fichier);
            break;
      }
   }
   while ((choix!='S') && (choix!='s'));
}
```

COURS et TP DE LANGAGE C++

Chapitre 10

Programmation orientée objet

Joëlle MAILLEFERT

joelle.maillefert@iut-cachan.u-psud.fr

IUT de CACHAN

Département GEII 2

CHAPITRE 10

PROGRAMMATION ORIENTE OBJET : NOTION DE CLASSE

I- INTRODUCTION

On attend d'un programme informatique :
- l'exactitude (réponse aux spécifications)
- la robustesse (réaction correcte à une utilisation « hors normes »)
- l'extensibilité (aptitude à l'évolution)
- la réutilisabilité (utilisation de modules)
- la portabilité (support d'une autre implémentation)
- l'efficience (performance en termes de vitesse d'exécution et de consommation mémoire)

Les langages évolués de type C ou PASCAL, reposent sur le principe de la programmation structurée (algorithmes + structures de données)
Le C++ et un langage orienté objet. Un langage orienté objet permet la manipulation de *classes*. Comme on le verra dans ce chapitre, la classe généralise la notion de structure.
Une classe contient des variables (ou « données ») et des fonctions (ou « méthodes ») permettant de manipuler ces variables.
Les langages « orientés objet » ont été développés pour faciliter l'écriture et améliorer la qualité des logiciels en termes de modularité et surtout de réutilisation.
Un langage orienté objet est livré avec une bibliothèque de classes. Le développeur utilise ces classes pour mettre au point ses logiciels.

Rappel sur la notion de prototype de fonction:

En C++, comme en C, on a fréquemment besoin de déclarer des *prototypes* de fonctions.
En particulier, on trouve dans les fichiers d'en-tête (de type *.h), des *prototypes* de fonctions appelées par le programme.
Le *prototype* d'une fonction est constitué du nom de la fonction, du type de la valeur de retour, du type des arguments à passer

Exemples: **void ma_fonction1()**

```
void ma_fonction2(int n, float u)   // prototype « complet »
void ma_fonction2(int, float)       // prototype « réduit »

int ma_fonction3(char *x)           // prototype « complet »
int ma_fonction3(char *)            // prototype « réduit »

int ma_fonction4(int &u)            // prototype « complet »
int ma_fonction4(int &)             // prototype « réduit »
```

On utilise indifféremment, dans les fichiers d'en-tête, le prototype complet ou le prototype réduit. Il est recommandé, pour une meilleure lisibilité, d'utiliser le prototype complet.

II- NOTION DE CLASSE

Exemple (à tester) et exercice X-1 :
(il faut ajuster la temporisation aux performances de la machine utilisée)

```
#include <iostream.h> // les classes
#include <conio.h>

    class point
   {
   private:
      int x,y;
   public:
      void initialise(int abs, int ord);
      void deplace(int abs, int ord);
      void affiche();
   };

    void point::initialise(int abs,int ord)
   {x = abs; y = ord;}

    void point::deplace(int dx,int dy)
   {x = x+dx; y = y+dy;}

    void point::affiche()
   {gotoxy(x,y);cout<<"Je suis en "<< x <<"  "<< y <<"\n";}

    void tempo(int duree)
   {
      float stop ;stop = duree*10000.0;
      for(;stop>0;stop=stop-1.0);}
// on utilise un float pour obtenir une temporisation suffisamment longue

    void main()
   {
      point a,b;
      a.initialise(1,4);
      a.affiche();
      tempo(10);
      a.deplace(17,10);
      a.affiche();
      b = a;          // affectation autorisée
      tempo(15);
      clrscr();
      b.affiche();
      getch() ;
   }
```

« point » est une ***classe***. Cette classe est constituée des ***données*** *privées x* et y et des ***fonctions membres*** **publiques (*ou méthodes*) «** initialise », « deplace », « affiche ». On déclare la classe en début de programme (données et prototype des fonctions membres), puis on définit le contenu des fonctions membres.
Les données x et y sont dites ***privées.*** Ceci signifie que l'on ne peut les manipuler qu'au travers des fonctions membres publiques. On dit que le langage C++ réalise ***l'encapsulation des données.***
a et b sont des ***objets*** de classe «point», c'est-à-dire des variables de type «point».
On a défini ici un nouveau type de variable, propre à cet application, comme on le fait en C avec les structures.

Suivant le principe dit de « l'encapsulation des données », la notation **a.x** est interdite à l'extérieur de la classe.

Exercice X-2:

Utiliser la classe « point » précédente. Ecrire une fonction de prototype **void test()** dans laquelle on déclare un point u, on l'initialise, on l'affiche , on le déplace et on l'affiche à nouveau. Le programme principal ne contient que l'appel à **test**.

Exercice X-3:

Ecrire une fonction de prototype **void test(point &u)** (référence) similaire. Ne pas déclarer de point local dans **test**. Déclarer un point local a dans le programme principal et appeler la fonction **test** en passant le paramètre a.

Exercice X-4:

Ecrire une fonction de prototype **point test()** qui retourne un point. Ce point sera initialisé et affiché dans **test** puis déplacé et à nouveau affiché dans le programme principal.

III- NOTION DE CONSTRUCTEUR

Un constructeur est une fonction membre *systématiquement exécutée* lors de la déclaration d'un objet statique, automatique, ou dynamique.

On ne traitera dans ce qui suit que les objets automatiques.

Dans l'exemple de la classe **point**, le constructeur remplace la fonction membre **initialise.**

Exemple (à tester) et exercice X-5:

```
#include <iostream.h> // notion de constructeur
#include <conio.h>

    class point
   {
      int x,y;
   public:
      point();
      // noter le prototype du constructeur (pas de "void")
      void deplace(int dx,int dy);
      void affiche();
   };

    point::point() // initialisation sans paramètre
   {
      x = 20; y = 10;
   }  // grace au constructeur

    void point::deplace(int dx,int dy)
   {
      x = x+dx;
      y = y+dy;
   }

    void point::affiche()
   {
      gotoxy(x,y);
      cout<<"Je suis en "<< x <<"  "<< y <<"\n";
   }

    void tempo(int duree)
   {
      float stop ;stop = duree*10000.0;
      for(;stop>0;stop=stop-1.0);
   }

    void main()
   {
      point a,b; // les deux points sont initialisés en 20,10
      a.affiche();
      tempo(10);
      a.deplace(17,10);
      a.affiche();
      tempo(15);
      clrscr();
      b.affiche();
      getch() ;
   }
```

Exemple (à tester) et exercice X-6:

```
#include <iostream.h> // introduction au constructeur
#include <conio.h>

    class point
   {
      int x,y;
   public:
      point(int abs,int ord);
    // noter l'absence de type de retour du constructeur (pas de "void")
      void deplace(int dx,int dy);
      void affiche();
   };

    point::point(int abs,int ord) // initialisation avec paramètres
   {  // grâce au constructeur, ici paramètres à passer
      x = abs; y = ord;
   }

    void point::deplace(int dx,int dy)
   {
      x = x+dx; y = y+dy;
   }

    void point::affiche()
   {
      gotoxy(x,y);cout<<"Je suis en "<< x <<"  "<< y <<"\n";
   }

    void tempo(int duree)
   {
      float stop = duree*10000.0;
      for(;stop>0;stop=stop-1.0);
   }

    void main()
   {  // les deux points sont initialisés : a en (20,10)  b en (30,20)
      point a(20,10),b(30,20);
      a.affiche(); tempo(10);
      a.deplace(17,10);
      a.affiche(); tempo(15);
      clrscr(); b.affiche();
      getch();
   }
```

Exercice X-7: Reprendre l'exercice X-2, en utilisant la classe de l'exercice X-6

Exercice X-8: Reprendre l'exercice X-3, en utilisant la classe de l'exercice X-6

Exercice X-9: Reprendre l'exercice X-4, en utilisant la classe de l'exercice X-6

IV- NOTION DE DESTRUCTEUR

Le destructeur est une fonction membre *systématiquement exécutée* (si elle existe !) «à la fin de la vie » d'un objet statique, automatique, ou dynamique.
On ne peut pas passer de paramètres au destructeur.
Sa tâche est de libérer la mémoire allouée à l'objet lors de sa création (via le constructeur). Si l'on ne définit pas de destructeur, il en existe un par défaut, non visible dans le code, qui assure la même fonction de libération de la mémoire.

On ne traitera dans ce qui suit que les objets automatiques.

Exemple (à tester) et exercice X-10:

```
#include <iostream.h> // notion de destructeur
#include <conio.h>

    class point
   {
      int x,y;
   public:  point(int abs,int ord);
      void deplace(int dx,int dy);
      void affiche();
      ~point(); // noter la convention adoptée pour le destructeur
   };

    point::point(int abs,int ord)
    // initialisation à partir de paramètres formels
   {
      x = abs; y = ord; // grâce au constructeur, ici paramètres à passer
   }

    void point::deplace(int dx,int dy)
   {
      x = x+dx; y = y+dy;
   }

    void point::affiche()
   {
      gotoxy(x,y);cout<<"Je suis en "<< x <<"  "<< y <<"\n";
   }

    point::~point()
   {
      cout<<"Frapper une touche..."; getch();
      cout<<"destruction du point x ="<< x <<" y="<< y <<"\n";
   }

    void tempo(int duree)
   {
      float stop = duree*10000.0;
      for(;stop>0;stop=stop-1.0);
   }

    void test()
   {
      point u(3,7); u.affiche(); tempo(20);
   }

    void main()
   {
      point a(1,4); a.affiche(); tempo(20);
      test();
      point b(5,10); b.affiche(); getch() ;
   }
```

V- ALLOCATION DYNAMIQUE

Lorsque les membres données d'une classe sont des pointeurs, le constructeur est utilisé pour l'allocation dynamique de mémoire sur ce pointeur.
Pour certaines applications, on n'est pas capable de définir la taille d'un tableau au moment de la compilation (exemple : tableau de mesures). La taille effective est fonction d'un contexte d'éxécution. En conséquence, une allocation dynamique s'impose. Les éléments du tableau seront accessibles via un pointeur qui sera une donnée membre privée.
Le destructeur est utilisé pour libérer la place. Il est obligatoire et doit être pris en charge par le programmeur.

Exemple (à tester) et exercice X-11:

```
#include <iostream.h> // Allocation dynamique de données membres
#include <stdlib.h>
#include <conio.h>

    class calcul
   {
   private :
      int nbval,*val;
   public:
      calcul(int nb,int mul);  // constructeur
      ~calcul();               // destructeur
      void affiche();
   };

    calcul::calcul(int nb,int mul)  //constructeur
   {
      nbval = nb;
      val = new int[nbval];  // réserve de la place en mémoire
      for(int i=0;i<nbval;i++)
      {
         val[i] = i*mul;
      }
   }

    calcul::~calcul()
   {
      delete val; // libération de la place réservée
   }

    void calcul::affiche()
   {
      for(int i=0;i<nbval;i++)
      {
         cout<< val[i] <<" ";
      }
      cout<<"\n";
   }

    void main()
   {
      clrscr();
      calcul suite1(10,4);
      suite1.affiche();
      calcul suite2(6,8);
      suite2.affiche();
      getch() ;
   }
```

VII- CORRIGE DES EXERCICES

Exercice X-2:

```
#include <iostream.h> // les classes
#include <conio.h>

    class point
   {
   private:
      int x,y;
   public:
      void initialise(int abs,int ord);
      void deplace(int dx,int dy);
      void affiche();
   };

    void point::initialise(int abs,int ord)
   {
      x = abs; y = ord;
   }

    void point::deplace(int dx,int dy)
   {
      x = x+dx; y = y+dy;
   }

    void point::affiche()
   {
      gotoxy(x,y);cout<<"Je suis en "<< x <<"  "<< y <<"\n";
   }

    void tempo(int duree)
   {
      float stop = duree*10000.0;
      for(;stop>0;stop=stop-1.0);
   }

    void test()
   {
      point u;
      u.initialise(1,4); u.affiche();
      tempo(10);
      u.deplace(17,10);  u.affiche();
   }

    void main()
   {
      test(); getch();
   }
```

Exercice X-3:

```
#include <iostream.h> // les classes
#include <conio.h>

    class point
   {
   private:
      int x,y;
   public:
      void initialise(int abs,int ord);
      void deplace(int,int);
      void affiche();
   };

    void point::initialise(int abs,int ord)
   {
      x = abs; y = ord;
   }

    void point::deplace(int dx,int dy)
   {
      x = x+dx; y = y+dy;
   }

    void point::affiche()
   {
      gotoxy(x,y);cout<<"Je suis en "<< x <<"  "<< y <<"\n";
   }

    void tempo(int duree)
   {
      float stop = duree*10000.0;
      for(;stop>0;stop=stop-1.0);
   }

    void test(point &u)
   {
      u.initialise(1,4);u.affiche();
      tempo(10);
      u.deplace(17,10);u.affiche();
   }

    void main()
   {
      point a;
      test(a);
      getch() ;
   }
```

Exercice X-4:

```
#include <iostream.h> // les classes
#include <conio.h>

    class point
   {
   private :
      int x,y;
   public:
      void initialise(int abs,int ord);
      void deplace(int dx,int dy);
      void affiche();
   };

    void point::initialise(int abs,int ord)
   {
      x = abs; y = ord;
   }

    void point::deplace(int dx,int dy)
   {
      x = x+dx; y = y+dy;
   }

    void point::affiche()
   {
      gotoxy(x,y);cout<<"Je suis en "<< x <<"  "<< y <<"\n";
   }

    void tempo(int duree)
   {
      float stop = duree*10000.0;
      for(;stop>0;stop=stop-1.0);
   }

    point test()
   {
      point u;
      u.initialise(1,4);u.affiche();
      return u;
   }

    void main()
   {
      point a;
      a = test();
      tempo(10);
      a.deplace(17,10);
      a.affiche(); getch() ;
   }
```

Exercice X-7:

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

    class point
   {
   private:
      int x,y;
   public:
      point(int abs,int ord);
    // noter l'absence de type de retour du constructeur (pas de "void")
      void deplace(int dx,int dy);
      void affiche();
   };

    point::point(int abs,int ord) // initialisation par des paramètres
   {
      x = abs; y = ord; // grâce au constructeur, ici paramètres à passer
   }

    void point::deplace(int dx,int dy)
   {
      x = x+dx; y = y+dy;
   }

    void point::affiche()
   {
      gotoxy(x,y);cout<<"Je suis en "<< x <<"  "<< y <<"\n";
   }

    void tempo(int duree)
   {
      float stop = duree*10000.0;
      for(;stop>0;stop=stop-1.0);
   }

    void test()
   {
      point u(1,4);
      u.affiche();
      tempo(10);
      u.deplace(17,10);
      u.affiche();
   }

    void main()
   {
      test();
      getch();
   }
```

Exercice X-8:

```
#include <iostream.h>   // les classes
#include <conio.h>

    class point
   {
   private :
      int x,y;
   public:
      point(int abs,int ord);
    // noter l'absence du type de retour du constructeur (pas de "void")
      void deplace(int dx,int dy);
      void affiche();
   };

    point::point(int abs,int ord) // initialisation par des paramètres
   {
      x = abs; y = ord; // grâce au constructeur, ici paramètres à passer
   }

    void point::deplace(int dx,int dy)
   {
      x = x+dx; y = y+dy;
   }

    void point::affiche()
   {
      gotoxy(x,y);cout<<"Je suis en "<< x <<"  "<< y <<"\n";
   }

    void tempo(int duree)
   {
      float stop = duree*10000.0;
      for(;stop>0;stop=stop-1.0);
   }

    void test(point &u)
   {
      u.affiche();
      tempo(10);
      u.deplace(17,10);
      u.affiche();
   }

    void main()
   {
      point a(1,4);
      test(a);
      getch() ;
   }
```

Exercice X-9:

```
#include <iostream.h> // les classes
#include <conio.h>

    class point
   {
   private:
      int x,y;
   public:
      point(int abs,int ord);
    // noter l'absence de type de retour du constructeur (pas de "void")
      void deplace(int dx,int dy);
      void affiche();
   };

    point::point(int abs,int ord) // initialisation par des paramètres
   {
      x = abs; y = ord; // grâce au constructeur, ici paramètres à passer
   }

    void point::deplace(int dx,int dy)
   {
      x = x+dx; y = y+dy;
   }

    void point::affiche()
   {
      gotoxy(x,y);cout<<"Je suis en "<< x <<"  "<< y <<"\n";
   }

    void tempo(int duree)
   {
      float stop = duree*10000.0;
      for(;stop>0;stop=stop-1.0);
   }

    point test()
   {
      point u(5,6); u.affiche();
      return u;
   }

    void main()
   {
      point a(1,4);
      a.affiche(); tempo(15);
      a = test();  tempo(10);
      a.deplace(17,10); a.affiche();
   }
```

COURS et TP DE LANGAGE C++

Chapitre 11

Propriétés des fonctions membres

Joëlle MAILLEFERT

joelle.maillefert@iut-cachan.u-psud.fr

IUT de CACHAN

Département GEII 2

CHAPITRE 11

PROPRIETES DES FONCTIONS MEMBRES

I- SURDEFINITION DES FONCTIONS MEMBRES

En utilisant la propriété de surcharge des fonctions du C++, on peut définir plusieurs constructeurs, ou bien plusieurs fonctions membres, différentes, mais portant le même nom.

Exemple (à tester) et exercice XI-1: Définition de plusieurs constructeurs:
Indiquer quel constructeur est utilisé pour chacun des objets a, b, c.

```
#include <iostream.h> // Surcharge de fonctions
#include <conio.h>

    class point
   {
   private:
      int x,y;
   public:
      point();    // constructeur 1
      point(int abs);  // constructeur 2
      point(int abs,int ord);  // constructeur 3
      void affiche();
   };

    point::point()  // constructeur 1
   {
      x=0; y=0;
   }

    point::point(int abs)  // constructeur 2
   {
      x = abs; y = abs;
   }

    point::point(int abs,int ord)  // constructeur 3
   {
      x = abs; y = ord;
   }

    void point::affiche()
   {
      gotoxy(x,y);cout<<"Je suis en "<<x<<" "<<y<<"\n";
   }
```

```
void main()
{
   point a, b(5), c(3,12);
   clrscr(); a.affiche(); b.affiche(); c.affiche(); getch();
}
```

Exercice XI-2: Surcharge d'une fonction membre
Écrire une deuxième fonction **affiche** de prototype **void point::affiche(char *message)**
Cette fonction donne la possibilité à l'utilisateur d'ajouter, à la position du point, un message sur l'écran.

II- FONCTIONS MEMBRES " EN LIGNE "

Le langage C++ autorise la description des fonctions membres dès leur déclaration dans la classe. On dit que l'on écrit une fonction " inline ".
Il s'agit alors d'une " macrofonction ": A chaque appel, le compilateur génère le code de la fonction et ne fait pas appel à un sous-programme.
Les durées d'exécution sont donc plus rapides mais cette méthode génère davantage de code.

Exemple (à tester) et exercice XI-3:

Comparer la taille des fichiers exXI_1.obj et exXI_3.obj

```
#include <iostream.h> // Fonctions en ligne
#include <conio.h>

    class point
   {
   private:
      int x,y;
   public:
       point()
      {
         x=0; y=0; // constructeur 1
      }
       point(int abs)
      {
         x=abs; y=abs; // constructeur 2
      }
       point(int abs,int ord)
      {
         x=abs; y=ord; // constructeur 3
      }
      void affiche();
   };
```

```
 void point::affiche()
{
   gotoxy(x,y);cout<<"Je suis en "<<x<<" "<<y<<"\n";
}

 void main()
{
   point a,b(5),c(3,12);
   a.affiche();
   b.affiche();
   c.affiche();
   getch() ;
}
```

III- INITIALISATION DE PARAMETRES PAR DEFAUT

Exemple (à tester) et exercice XI-4:

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

    class point
   {
   private:
      int x,y;
   // Fonctions membres " en ligne "
   public: // arguments par défaut
       point(int abs=0,int ord=2)
      {
         x=abs; y=ord; // constructeur
      }
      void affiche(char *message = "Position du point");
   };

    void point::affiche(char *message)
   {
      gotoxy(x,y-1); cout<<message;
      gotoxy(x,y);   cout<<"Je suis en "<<x<<" "<<y<<"\n";
   }

    void main()
   {
      point a, b(40), c(3,12);
      char texte[10]="Bonjour";
      a.affiche();
      b.affiche("Point b");
      c.affiche(texte);
      getch();
   }
```

IV- OBJETS TRANSMIS EN ARGUMENT D'UNE FONCTION MEMBRE

Quand on passe comme paramètre à une fonction membre un objet de la classe à laquelle appartient cette fonction:

Exemple (à tester) et exercice XI-5:

```
#include <iostream.h>
#include <conio.h>
// objets transmis en argument d'une fonction membre
    class point
   {
   private:
      int x,y;
   public:
       point(int abs = 0,int ord = 2)
      {
         x=abs; y=ord;// constructeur
      }
      int coincide(point pt);
   };

    int point::coincide(point pt)
   {  // noter la dissymétrie des notations pt.x et x
      if ((pt.x == x) && (pt.y == y))
         return(1);
      else
         return(0);
   }

    void main()
   {
      int test1,test2;
      point a,b(1),c(0,2);
      test1 = a.coincide(b);
      test2 = b.coincide(a);
      cout<<"a et b:"<<test1<<" ou "<<test2<<"\n";
      test1 = a.coincide(c);
      test2 = c.coincide(a);
      cout<<"a et c:"<<test1<<" ou "<<test2<<"\n";
      getch() ;
   }
```

Noter que l'on rencontre la notation " pt.x " ou " pt.y " pour la première fois. Elle n'est autorisée qu'à l'intérieur d'une fonction membre (x et y membres privés de la classe).

On verra plus tard que le passage d'un objet par valeur pose problème si certains membres de la classe sont des pointeurs. Il faudra alors prévoir une allocation dynamique de mémoire via un constructeur.

Exercice XI-6: On définit la classe **vecteur** comme ci-dessous:

```
 class vecteur
{
private:
   float x,y;
public:
   vecteur(float abs,float ord);
   void homothetie(float val);
   void affiche();
};

 vecteur::vecteur(float abs =0.,float ord = 0.)
{
   x=abs; y=ord;
}

 void vecteur::homothetie(float val)
{
   x = x*val;  y = y*val;
}

 void vecteur::affiche()
{
   cout<<"x = "<<x<<"  y = "<<y<<"\n";
}
```

La mettre en oeuvre dans **void main(),** en ajoutant une fonction membre **float det(vecteur)** qui retourne le déterminant des deux vecteurs (celui passé en paramètre et celui de l'objet).

V- OBJET RETOURNE PAR UNE FONCTION MEMBRE

Que se passe-t-il lorsqu'une fonction membre retourne elle-même un objet ?

1- Retour par valeur

Exemple (à tester) et exercice XI-7: (la fonction concernée est la fonction **symetrique**)

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

// La valeur de retour d'une fonction est un objet
// Retour par valeur

    class point
   {
   private:
      int x,y;
   public:
       point(int abs = 0,int ord = 0)
      {
         x=abs; y=ord; // constructeur
      }
      point symetrique();
      void affiche();
   };

    point point::symetrique()
   {
      point res;
      res.x = -x;  res.y = -y;
      return res;
   }

    void point::affiche()
   {
      cout<<"Je suis en "<<x<<" "<<" "<<y<<"\n";
   }

    void main()
   {
      point a,b(1,6);
      a=b.symetrique();
      a.affiche();b.affiche();
      getch();
   }
```

2- Retour par adresse (***)

Quand on a besoin de retourner une adresse.

Exemple (à tester) et exercice XI-8:

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

// La valeur de retour d'une fonction est l'adresse d'un objet

    class point
   {
   private:
      int x,y;
   public:
       point(int abs = 0,int ord = 0)
      {
         x=abs; y=ord; // constructeur
      }
      point *symetrique();
      void affiche();
   };

    point *point::symetrique()
   {
      point *res;
      res = new point;
      res->x = -x;  res->y = -y;
      return res;
   }

    void point::affiche()
   {
      cout<<"Je suis en "<<x<<" "<<y<<"\n";
   }

    void main()
   {
      point a,b(1,6);
      a = *b.symetrique();
      a.affiche();b.affiche();
      getch();
   }
```

3- Retour par référence (***)

La valeur retournée l'est par référence. On en verra l'usage dans un prochain chapitre.

Exemple (à tester) et exercice XI-9:

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

// La valeur de retour d'une fonction est la référence d'un objet

    class point
   {
   private:
      int x,y;
   public:
       point(int abs = 0,int ord = 0)
      {
         x=abs; y=ord; // constructeur
      }
      point &symetrique();
      void affiche();
   };

    point &point::symetrique()
   {  // La variable res est obligatoirement static
      // pour retourner par référence
      static point res;
      res.x = -x;  res.y = -y;
      return res;
   }

    void point::affiche()
   {
      cout<<"Je suis en "<<x<<" "<<y<<"\n";
   }

    void main()
   {
      point a,b(1,6);
      a=b.symetrique();
      a.affiche();b.affiche();
      getch();
   }
```

Remarque: " res " et " b.symetrique "occupent le même emplacement mémoire (car " res " est une référence à " b.symetrique ". On déclare donc " res " comme variable static, sinon, cet objet n'existerait plus après être sorti de la fonction.

Exercice XI-10: Reprendre la classe **vecteur.** Modifier la fonction **homotéthie**, qui retourne le vecteur modifié. (prototype: **vecteur vecteur::homotethie(float val)**).

Exercice XI-11 (***): Même exercice, le retour se fait par adresse.

Exercice XI-12 (***): Même exercice, le retour se fait par référence.

VI- LE MOT CLE " THIS "

Ce mot désigne l'adresse de l'objet invoqué. Il est *utilisable uniquement* au sein d'une fonction membre.

Exemple (à tester) et exercice XI-13:

```
#include <conio.h>
#include <iostream.h>
// le mot clé this : pointeur l'objet sur l'instance de point
// utilisable uniquement dans une fonction membre
    class point
   {
   private:
      int x,y;
   public:
       point(int abs=0, int ord=0) // constructeur en ligne
      {
         x=abs; y=ord;
      }
      void affiche();
   };

    void point::affiche()
   {
      cout<<"Adresse: "<<this<<" - Coordonnées: "<<x<<" "<<y<<"\n";
   }

    void main()
   {
      point a(5),b(3,15);
      a.affiche();b.affiche();
      getch();
   }
```

Exercice XI-14: Remplacer, dans l'exercice XI-6, la fonction **coïncide** par la fonction suivante:

```
    int point::coincide(point *adpt)
   {
      if ((this->x == adpt->x) && (this->y == adpt->y))
         return(1);
      else
         return(0);
   }
```

Exercice XI-15: Reprendre la classe **vecteur**, munie du constructeur et de la fonction d'affichage. Ajouter
- Une fonction membre **float vecteur::prod_scal(vecteur)** qui retourne le produit scalaire des 2 vecteurs.
- Une fonction membre **vecteur vecteur::somme(vecteur)** qui retourne la somme des 2 vecteurs.

VII- CORRIGE DES EXERCICES

Exercice XI-2:

```
#include <iostream.h> // Surcharge de fonctions
#include <conio.h>

    class point
   {
   private:
      int x,y;
   public:
      point();  // constructeur 1
      point(int abs);  // constructeur 2
      point(int abs,int ord);  // constructeur 3
      void affiche();
      void affiche(char *message); // argument de type chaîne
   };

    point::point()  // constructeur 1
   {
      x=0; y=0;
   }

    point::point(int abs)  // constructeur 2
   {
      x=y=abs;
   }

    point::point(int abs,int ord)  // constructeur 3
   {
      x = abs; y = ord;
   }

    void point::affiche() // affiche 1
   {
      gotoxy(x,y);cout<<"Je suis en "<<x<<" "<<y<<"\n";
   }

    void point::affiche(char *message) // affiche 2
   {
      gotoxy(x,y-1);cout<<message;
      gotoxy(x,y);cout<<"Je suis en "<<x<<" "<<y<<"\n";
   }

    void main()
   {
      point a,b(5),c(3,12); char texte[10] = "Bonjour";
      a.affiche(); b.affiche("Point b:"); c.affiche(texte); getch();
   }
```

Exercice XI-6:

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

// Classe vecteur - Fonction membre déterminant

    class vecteur
   {
   private:
      float x,y;
   public:
      vecteur(float abs,float ord);
      void homothetie(float val);
      void affiche();
      float det(vecteur v);
   };

    vecteur::vecteur(float abs =0.,float ord = 0.)
   {
      x=abs; y=ord;
   }

    void vecteur::homothetie(float val)
   {
      x = x*val;  y = y*val;
   }

    void vecteur::affiche()
   {
      cout<<"x = "<<x<<"  y = "<<y<<"\n";
   }

    float vecteur::det(vecteur w)
   {
      float res;
      res = x * w.y - y * w.x;
      return res;
   }

    void main()
   {
      vecteur v(2,6),u(4,8);
      v.affiche();
      v.homothetie(2);
      v.affiche();
      cout <<"Déterminant de (u,v) = "<<v.det(u)<<"\n";
      cout <<"Déterminant de (v,u) = "<<u.det(v)<<"\n";
      getch();
   }
```

Exercice XI-10:

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

// Classe vecteur - Fonction homothétie - Retour par valeur

    class vecteur
   {
   private:
      float x,y;
   public:
      vecteur(float abs,float ord); // Constructeur
      vecteur homothetie(float val);
      void affiche();
   };

    vecteur::vecteur(float abs =0,float ord = 0)
   {
      x=abs; y=ord;
   }

    vecteur vecteur::homothetie(float val)
   {
      vecteur res;
      res.x = x*val;  res.y = y*val;
      return res;
   }

    void vecteur::affiche()
   {
      cout<<"x = "<<x<<"  y = "<<y<<"\n";
   }

    void main()
   {
      vecteur v(2,6),u;
      v.affiche();
      u.affiche();
      u = v.homothetie(4);
      u.affiche();
      getch();
   }
```

Exercice XI-11:

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

// Classe vecteur - Fonction homothétie - Retour par adresse

    class vecteur
   {
   private:
      float x,y;
   public:
      vecteur(float abs,float ord); // Constructeur
      vecteur  *homothetie(float val);
      void affiche();
   };

    vecteur::vecteur(float abs =0.,float ord = 0.) // Constructeur
   {
      x=abs; y=ord;
   }

    vecteur *vecteur::homothetie(float val)
   {
      vecteur *res;
      res = new vecteur;
      res->x = x*val;  res->y = y*val;
      return res;
   }

    void vecteur::affiche()
   {
      cout<<"x = "<<x<<"  y = "<<y<<"\n";
   }

    void main()
   {
      vecteur v(2,6),u;
      v.affiche();
      u.affiche();
      u = *v.homothetie(4);
      u.affiche();
      getch();
   }
```

Exercice XI-12:

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

// Classe vecteur -Fonction homothétie - Retour par référence

    class vecteur
   {
   private:
      float x,y;
   public:
      vecteur(float abs,float ord); // Constructeur
      vecteur &homothetie(float val);
      void affiche();
   };

    vecteur::vecteur(float abs =0,float ord = 0)
   {
      x=abs; y=ord;
   }

    vecteur &vecteur::homothetie(float val)
   {
      static vecteur res;
      res.x = x*val;
      res.y = y*val;
      return res;
   }

    void vecteur::affiche()
   {
      cout<<"x = "<<x<<"  y = "<<y<<"\n";
   }

    void main()
   {
      vecteur v(2,6),u;
      v.affiche();
      u.affiche();
      u = v.homothetie(4);
      u.affiche();
      getch();
   }
```

Exercice XI-14:

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

// Objets transmis en argument d'une fonction membre
// Utilisation du mot clé this

    class point
   {
   private:
      int x,y;
   public:
       point(int abs = 0,int ord = 0)
      {
         x=abs; y=ord; // constructeur
      }
      int coincide(point *adpt);
   };

    int point::coincide(point *adpt)
   {
      if ((this->x == adpt->x) && (this->y == adpt->y))
         return(1);
      return(0);
   }

    void main()
   {
      point a,b(1),c(1,0);
      cout<<"a et b:"<<a.coincide(&b)<<" ou "<<b.coincide(&a)<<"\n";
      cout<<"b et c:"<<b.coincide(&c)<<" ou "<<c.coincide(&b)<<"\n";
      getch();
   }
```

Exercice XI-15:

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

// Création d'une classe vecteur, avec constructeur, affichage
// Produit scalaire de 2 vecteurs

    class vecteur
   {
   private:
      float x,y;
   public:
      vecteur(float xpos,float ypos);
      vecteur somme(vecteur v);
      float prod_scal(vecteur v);
      void affiche();
   };

    vecteur::vecteur(float xpos=0,float ypos=0)
   {
      x = xpos; y = ypos;
   }

    float vecteur::prod_scal(vecteur v)
   {  // tester le passage par référence &v
      float res = (x * v.x) + (y * v.y);
      return (res);
   }

    vecteur vecteur::somme(vecteur v)
   {  // tester aussi le passage par référence &v
      vecteur res;
      res.x = x + v.x; res.y = y + v.y;
      return res;
   }

    void vecteur::affiche()
   {
      cout<<"x= "<<x<<" y= "<<y<<"\n";
   }

    void main()
   {
      vecteur a(3);a.affiche(); vecteur b(1,2);b.affiche();
      vecteur c(4,5),d;c.affiche();
      cout<<"b.c = "<<b.prod_scal(c)<<"\n";
      cout<<"c.b = "<<c.prod_scal(b)<<"\n";
      c = a.somme(b); d = b.somme(a);
      cout<<"Coordonnées de a+b:";c.affiche();cout<<"\n";
      cout<<"Coordonnées de b+a:";d.affiche();cout<<"\n"; getch();
   }
```

COURS et TP DE LANGAGE C++

Chapitre 12

Initialisation, construction, destruction des objets

Joëlle MAILLEFERT

joelle.maillefert@iut-cachan.u-psud.fr

IUT de CACHAN

Département GEII 2

CHAPITRE 12 (***)

INITIALISATION, CONSTRUCTION, DESTRUCTION DES OBJETS

Dans ce chapitre, on va chercher à mettre en évidence les cas pour lesquels le compilateur cherche à exécuter un constructeur, et quel est ce constructeur et, d'une façon plus générale, on étudiera les mécanismes de construction et de destruction.

I- CONSTRUCTION ET DESTRUCTION DES OBJETS AUTOMATIQUES

Rappel: Une variable locale est appelée encore « automatique », si elle n'est pas précédée du mot « static ». Elle n'est alors pas initialisée et sa portée (ou durée de vie) est limitée au bloc où elle est déclarée.

Exemple et exercice XII-1:
Exécuter le programme suivant et étudier soigneusement à quel moment sont créés puis détruits les objets déclarés. Noter l'écran d'exécution obtenu.

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

    class point
   {
   private:
      int x,y;
   public:
      point(int abs,int ord);
      ~point();
   };

    point::point(int abs,int ord)
   {
      x = abs; y = ord;
      cout<<"Construction du point "<< x <<"  "<< y <<"\n";
   }

    point::~point()
   {cout<<"Destruction du point "<< x <<" "<< y <<"\n";}

    void test()
   {cout<<"Début de test()\n";point u(3,7);cout<<"Fin de test()\n";}

    void main()
   {
      cout<<"Début de main()\n";point a(1,4); test();
      point b(5,10);
      for(int i=0;i<3;i++)point(7+i,12+i);
      cout<<"Fin de main()\n"; getch();
   }
```

II- CONSTRUCTION ET DESTRUCTION DES OBJETS STATIQUES

Exemple et exercice XII-2: Même étude avec le programme suivant:

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

    class point
   {
   private :
      int x,y;
   public:
      point(int abs,int ord);
      ~point();
   };

    point::point(int abs,int ord)
   {
      x = abs; y = ord;
      cout<<"Construction du point "<< x <<"  "<< y <<"\n";
   }

    point::~point()
   {
      cout<<"Destruction du point "<< x <<" "<< y <<"\n";
   }

    void test()
   {
      cout<<"Début de test()\n";
      static point u(3,7); cout<<"Fin de test()\n";
   }

    void main()
   {
      cout<<"Début de main()\n";
      point a(1,4);
      test();
      point b(5,10);
      cout<<"Fin de main()\n";
      getch() ;
   }
```

III- CONSTRUCTION ET DESTRUCTION DES OBJETS GLOBAUX

Exemple et exercice XII-3: Même étude avec le programme suivant

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

    class point
   {
   private:
      int x,y;
   public:
      point(int abs,int ord);
      ~point();
   };

    point::point(int abs,int ord)
   {
      x = abs; y = ord;
      cout<<"Construction du point "<<x<<"  "<<y<<"\n";
   }

    point::~point()
   {
      cout<<"Destruction du point "<<x<<" "<<y<<"\n";
   }

   point a(1,4);  // variable globale

    void main()
   {
      cout<<"Début de main()\n";
      point b(5,10);
      cout<<"Fin de main()\n";
      getch();
   }
```

IV- CONSTRUCTION ET DESTRUCTION DES OBJETS DYNAMIQUES

Exemple et exercice XII-4: Même étude avec le programme suivant

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

    class point
   {
   private:
      int x,y;
   public:
      point(int abs,int ord);
      ~point();
   };

    point::point(int abs,int ord)
   {
      x = abs; y = ord;
      cout<<"Construction du point "<< x <<" "<< y <<"\n";
   }

    point::~point()
   {
      cout<<"Destruction du point "<< x <<" "<< y <<"\n";
   }

    void main()
   {
      cout<<"Début de main()\n";
      point *adr;
      adr = new point(3,7); // réservation de place en mémoire
      delete adr;  // libération de la place
      cout<<"Fin de main()\n";
      getch();
   }
```

Exécuter à nouveau le programme en mettant en commentaires l'instruction « delete adr ».

Donc, dans le cas d'un objet dynamique, le constructeur est exécuté au moment de la réservation de place mémoire (« new »), le destructeur est exécuté lors de la libération de cette place (« delete »).

V- INITIALISATION DES OBJETS

Le langage C++ autorise l'initialisation des variables dès leur déclaration:
Par exemple: **int i = 2;**

Cette initialisation est possible, et de façon plus large, avec les objets:
Par exemple: **point a(5,6); // constructeur avec arguments**

Et même: **point b = a;**

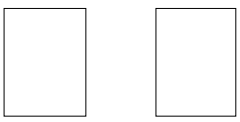
Que se passe-t-il alors à la création du point b ? En particulier, quel constructeur est-il exécuté?

Exemple et exercice XII-5: Tester l'exemple suivant, noter l'écran d'exécution obtenu et conclure

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

    class point
   {
   private :
      int x,y;
   public:
      point(int abs,int ord);
      ~point();}
   ;

    point::point(int abs,int ord)
   {
      x = abs; y = ord;
      cout<<"Construction du point "<< x <<" "<< y;
      cout<<" Son adresse: "<< this <<"\n";
   }

    point::~point()
   {
      cout<<"Destruction du point "<<x<<" "<<y;
      cout<<" Son adresse:"<<this<<"\n";
   }

    void main()
   {
      cout<<"Début de main()\n";
      point a(3,7);
      point b=a;
      cout<<"Fin de main()\n";
      clrscr();
   }
```

Sont donc exécutés ici:
- le constructeur pour a UNIQUEMENT
- le destructeur pour a ET pour b

Le compilateur affecte correctement des emplacements-mémoire différents pour a et b:

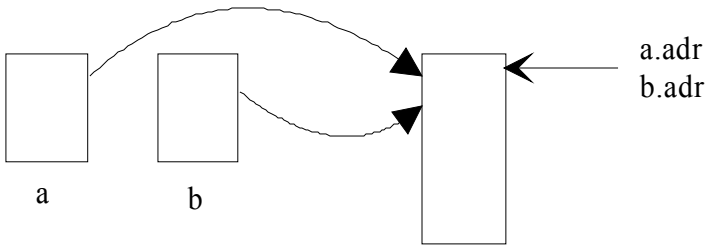


Exemple et exercice XII-6:
Dans l'exemple ci-dessous, la classe **liste** contient un membre privé de type pointeur. Le constructeur lui alloue dynamiquement de la place. Que se passe-t-il lors d'une initialisation de type: **liste a(3);**
liste b = a;

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

    class liste
   {
   private :
      int taille;
      float *adr;
   public:
      liste(int t);
      ~liste();
   };

    liste::liste(int t)
   {
      taille = t;adr = new float[taille]; cout<<"Construction";
      cout<<" Adresse de l'objet:"<<this;
      cout<<" Adresse de liste:"<< adr <<"\n";
   }

    liste::~liste()
   {
      cout<<"Destruction Adresse de l'objet:"<<this;
      cout<<" Adresse de liste:"<< adr <<"\n"; delete adr;
   }

    void main()
   {
      cout<<"Début de main()\n";
      liste a(3);
      liste b=a;
      cout<<"Fin de main()\n"; getch();
   }
```

Comme précédemment, sont exécutés ici:
- le constructeur pour a UNIQUEMENT
- le destructeur pour a ET pour b

Le compilateur affecte des emplacements-mémoire différents pour a et b.
Par contre, les pointeurs **b.adr** et **a.adr** pointent sur la même adresse. La réservation de place dans la mémoire ne s'est pas exécutée correctement:



Exercice XII-7:

Ecrire une fonction membre **void saisie()** permettant de saisir au clavier les composantes d'une liste et une fonction membre **void affiche()** permettant de les afficher sur l'écran. Les mettre en oeuvre dans **void main()** en mettant en évidence le défaut vu dans l'exercice IV-6.

L'étude de ces différents exemples montre que, lorsque le compilateur ne trouve pas de constructeur approprié, il exécute un constructeur par défaut,invisible du programmeur, dont la fonction est de copier les données non allouées dynamiquement .

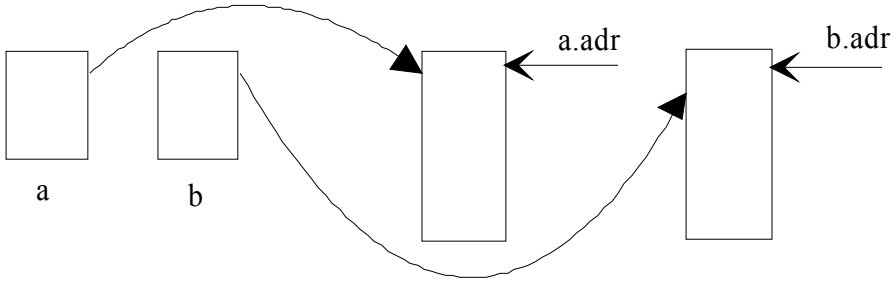
Exemple et exercice XII-8:

On va maintenant ajouter un constructeur de prototype **liste(liste &)** appelé encore « constructeur par recopie ». Ce constructeur sera appelé lors de l'exécution de **liste b=a;**

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

    class liste
   {
   private:
      int taille;
      float *adr;
   public:
      liste(int t);
      liste(liste &v);
      ~liste();
   };

    liste::liste(int t)
   {
      taille = t;adr = new float[taille];
      cout<<"\nConstruction"; cout<<"\nAdresse de l'objet:"<<this;
      cout<<"  Adresse de liste:"<<adr<<"\n";
   }

    liste::liste(liste &v)  // passage par référence obligatoire
   {
      taille = v.taille; adr = new float[taille];
      for(int i=0;i<taille;i++)adr[i] = v.adr[i];
      cout<<"\nConstructeur par recopie";
      cout<<"  Adresse de l'objet:"<<this;
      cout<<"  Adresse de liste:"<<adr<<"\n";
   }

    liste::~liste()
   {
      cout<<"\nDestruction Adresse de l'objet:"<<this;
      cout<<"  Adresse de liste:"<<adr<<"\n"; delete adr;
   }
```

```
 void main()
{
   cout<<"Debut de main()\n";
   liste a(3);
   liste b=a;
   cout<<"\nFin de main()\n";getch() ;
}
```

Ici, toutes les réservations de place en mémoire ont été correctement réalisées:



Exercice XII-9:

Reprendre l'exercice IV-7, et montrer qu'avec le « constructeur par recopie », on a résolu le problème rencontré.

VI- CONCLUSION

Il faut prévoir un « constructeur par recopie » lorsque la classe contient des données dynamiques.
Lorsque le compilateur ne trouve pas ce constructeur, aucune erreur n'est générée, par contre, le programme ne fonctionne pas.

VII - ROLE DU CONSTRUCTEUR LORSQU'UNE FONCTION RETOURNE UN OBJET

On va étudier maintenant une autre application du « constructeur par recopie ».

Exemple et exercice XII-10:

On reprend la fonction membre **point symetrique()** étudiée dans l'exercice III-11. Cette fonction retourne donc un objet.
Tester le programme suivant et étudier avec précision à quel moment les constructeurs et le destructeur sont exécutés.

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

    class point
   {
   private:
      int x,y;
   public:
      point(int abs,int ord);
      point(point &); // constructeur par recopie
      point symetrique();
       void affiche()
      { // Fonction inline
         cout<<"x="<< x <<" y="<< y <<"\n";
      }
      ~point();
   };

    point::point(int abs=0,int ord=0)
   {
      x = abs; y = ord; cout<<"Construction du point "<< x <<"  "<< y;
      cout<<" d'adresse "<< this <<"\n";
   }

    point::point(point &pt)
   {
      x = pt.x; y = pt.y;
      cout<<"Construction par recopie du point "<< x <<" "<< y;
      cout<<" d'adresse "<< this <<"\n";
   }

    point point::symetrique()
   {
      point res; res.x = -x; res.y = -y;
      return res;
   }

    point::~point()
   {
      cout<<"Destruction du point "<<x<<" "<<y;
      cout<<" d'adresse "<< this <<"\n";
   }
```

```
 void main()
{
   cout<<"Début de main()\n";
   point a(1,4), b;
   cout<<"Avant appel à symetrique\n";
   b = a.symetrique();
   b.affiche();
   cout<<"Après appel à symetrique et fin de main()\n"; getch() ;
}
```

Il y a donc création d'un objet temporaire, au moment de la transmission de la valeur de « res » à « b ». Le constructeur par recopie et le destructeur sont exécutés.
Il faut insister sur le fait que la présence du construteur par recopie n'était pas obligatoire ici: l'exercice III-1 a fonctionné correctement ! et se rappeler ce qui a été mentionné plus haut:

Lorsqu'un constructeur approprié existe, il est exécuté. S'il n'existe pas, aucune erreur n'est générée.
Il faut toujours prévoir un constructeur par recopie lorsque l'objet contient une partie dynamique.

Tester éventuellement le programme IV-10, en supprimant le constructeur par recopie.

Exemple et exercice XII-11:

On a écrit ici, pour la classe **liste** étudiée précédemment, une fonction membre de prototype **liste oppose()** qui retourne la liste de coordonnées opposées.
Exécuter ce programme et conclure.

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

 class liste
{
private :
   int taille;
   float *adr;
public:
   liste(int t);
   liste(liste &v);
   void saisie();
   void affiche();
   liste oppose();
   ~liste();
};
```

```
 liste::liste(int t)
{
   taille = t; adr = new float[taille];
   cout<<"Construction";
   cout<<" Adresse de l'objet:"<<this;
   cout<<" Adresse de liste:"<<adr<<"\n";
}

 liste::liste(liste &v)  // passage par référence obligatoire
{
   taille = v.taille;
   adr = new float[taille];
   for(int i=0;i<taille;i++)adr[i]=v.adr[i];
   cout<<"Constructeur par recopie";
   cout<<" Adresse de l'objet:"<< this;
   cout<<" Adresse de liste:"<< adr <<"\n";
}

 liste::~liste()
{
   cout<<"Destruction Adresse de l'objet:"<< this;
   cout<<" Adresse de liste:"<< adr <<"\n"; delete adr;
}

 void liste::saisie()
{
   for(int i=0;i<taille;i++)
   {cout<<"Entrer un nombre:"; cin>>*(adr+i);}
}

 void liste::affiche()
{
   for(int i=0;i<taille;i++)cout<<*(adr+i)<<" ";
   cout<<"adresse de l'objet: "<< this;
   cout<<" adresse de liste: "<< adr <<"\n";
}

 liste liste::oppose()
{
   liste res(taille);
   for(int i=0;i<taille;i++)res.adr[i] = - adr[i];
   for(i=0;i<taille;i++)cout<<res.adr[i]<<" ";
   cout<<"\n";
   return res;
}

 void main()
{
   cout<<"Début de main()\n";
   liste a(3), b(3);
   a.saisie();     a.affiche();
   b = a.oppose(); b.affiche();
   cout<<"Fin de main()\n"; getch();
}
```

Solution et exercice XII-12:

On constate donc que l'objet local **res** de la fonction **oppose()** est détruit AVANT que la transmission de valeur ait été faite. Ainsi, la libération de place mémoire a lieu trop tôt. Ré-écrire la fonction **oppose()** en effectuant le retour par référence (cf chapitre 3) et conclure sur le rôle du retour par référence.

VIII- EXERCICES RECAPITULATIFS

Exercice XII-13:

Ecrire une classe **pile_entier** permettant de gérer une pile d'entiers, selon le modèle ci-dessous.

```
 class  pile_entier
{
private:
 // pointeur de pile, taille maximum, hauteur courante
   int *pile,taille,hauteur;
public:
// constructeur : alloue dynamiquement de la mémoire
// taille de la pile(20 par défaut), initialise la hauteur à 0
   pile_entier(int n);
   ~pile_entier();     // destructeur
   void empile(int p); // ajoute un élément
   int depile(); // retourne la valeur de l'entier en haut de la pile
                 // la hauteur  diminue d'une unité
   int pleine(); // retourne 1 si la pile est pleine, 0 sinon
   int vide();   // retourne 1 si la pile est vide, 0 sinon
};
```

Mettre en oeuvre cette classe dans main(). Le programme principal doit contenir les déclarations suivantes:

```
 void main()
{
   pile_entier a,b(15); // pile automatique
   pile_entier *adp;    // pile dynamique
}
```

Exercice XII-14:

Ajouter un constructeur par recopie et le mettre en oeuvre.

IX- LES TABLEAUX D'OBJETS

Les tableaux d'objets se manipulent comme les tableaux classiques du langage C++
Avec la classe **point** déjà étudiée on pourra par exemple déclarer:

point courbe[100]; // déclaration d'un tableau de 100 points

La notation **courbe[i].affiche()** a un sens.
La classe courbe étant un tableau contenant des objets de la classe **point** doit dans ce cas, OBLIGATOIREMENT posséder un **constructeur** sans argument (ou avec tous les arguments avec valeur par défaut). *Le constructeur est exécuté pour chaque élément du tableau.*
La notation suivante est admise:

```
 class point
{
private:
   int x,y;
public:
    point(int abs=0,int ord=0)
   {
      x = abs;
      y = ord;
   }
};

 void main()
{
   point courbe[5] = {7,4,2};
}
```

On obtiendra les résultats suivants:

	x	y
courbe[0]	**7**	**0**
courbe[1]	**4**	**0**
courbe[2]	**2**	**0**
courbe[3]	**0**	**0**
courbe[4]	**0**	**0**

On pourra de la même façon créer un tableau dynamiquement:

point *adcourbe = new point[20];

et utiliser les notations ci-dessus. Pour détruire ce tableau, on écrira **delete []adcourbe;**
Le destructeur sera alors exécuté pour chaque élément du tableau.

Exercice XII-15:

Reprendre par exemple l'exercice III-8 (classe **vecteur**), et mettre en oeuvre dans le programme principal un tableau de vecteurs.

X - CORRIGE DES EXERCICES

Exercice XII-7:

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

    class liste
   {
   private:
      int taille;
      float *adr;
   public:
      liste(int t);
      void saisie();
      void affiche();
      ~liste();
   };

    liste::liste(int t)
   {
      taille = t;adr = new float[taille];cout<<"Construction";
      cout<<" Adresse de l'objet:"<<this;
      cout<<" Adresse de liste:"<<adr<<"\n";
   }

    liste::~liste()
   {
      cout<<"Destruction Adresse de l'objet:"<<this;
      cout<<" Adresse de liste:"<<adr<<"\n";
      delete adr;
   }

    void liste::saisie()
   {
      for(int i=0;i<taille;i++)
      {cout<<"Entrer un nombre:"; cin>>*(adr+i);}
   }

    void liste::affiche()
   {
      for(int i=0;i<taille;i++)cout<<*(adr+i)<<" ";
      cout<<"\n";
   }

    void main()
   {
      cout<<"Debut de main()\n";
      liste a(3);
      liste b=a;
      a.saisie();a.affiche();
      b.saisie();b.affiche();a.affiche();
      cout<<"Fin de main()\n"; getch() ;
   }
```

Exercice XII-9:

Même programme qu'au IV-7, en ajoutant le « constructeur par recopie » du IV-8.

Exercice XII-12:

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

    class liste
   {
   private:
      int taille;
      float *adr;
   public:
      liste(int t);
      liste(liste &v);
      void saisie();
      void affiche();
      liste &oppose();
      ~liste();
   };

    liste::liste(int t)
   {
      taille = t; adr = new float[taille];
      cout<<"Construction";
      cout<<" Adresse de l'objet:"<<this;
      cout<<" Adresse de liste:"<<adr<<"\n";
   }

    liste::liste(liste &v)  // passage par référence obligatoire
   {
      taille = v.taille;
      adr = new float[taille];
      for(int i=0;i<taille;i++)adr[i]=v.adr[i];
      cout<<"Constructeur par recopie";
      cout<<" Adresse de l'objet:"<<this;
      cout<<" Adresse de liste:"<<adr<<"\n";
   }

    liste::~liste()
   {
      cout<<"Destruction Adresse de l'objet:"<<this;
      cout<<" Adresse de liste:"<<adr<<"\n";
      delete adr;
   }

    void liste::saisie()
   {
      for(int i=0;i<taille;i++)
      {cout<<"Entrer un nombre:";cin>> *(adr+i) ;}
   }
```

```
 void liste::affiche()
{
   for(int i=0;i<taille;i++)cout<<*(adr+i)<<" ";
   cout<<"Adresse de l'objet: "<<this;
   cout<<" Adresse de liste: "<<adr<<"\n";
}

 liste &liste::oppose()
{
   static liste res(taille);
   for(int i=0;i<taille;i++)*(res.adr+i) = - *(adr+i);
   for(i=0;i<taille;i++)cout<<*(res.adr+i);
   cout<<"\n";
   return res;
}

 void main()
{
   cout<<"Début de main()\n";
   liste a(3),b(3);
   a.saisie();a.affiche();
   b = a.oppose();b.affiche();
   cout<<"Fin de main()\n";
   getch();
}
```

Exercice XII-13:

```
#include <iostream.h>  // Gestion d'une pile d'entiers
#include <conio.h>

 class  pile_entier
{
private :
   int *pile;
   int taille;
   int hauteur;
public:
   pile_entier(int n); // constructeur, taille de la pile
   ~pile_entier();     // destructeur
   void empile(int p); // ajoute un élément
   int depile();       // dépile un élément
   int pleine();       // 1 si vrai 0 sinon
   int vide();         // 1 si vrai 0 sinon
};
```

```
 pile_entier::pile_entier(int n=20)  // taille par défaut: 20
{
   taille = n;
   pile = new int[taille];  // taille de la pile
   hauteur = 0;
   cout<<"On a fabrique une pile de "<<taille<<" éléments\n";
}

 pile_entier::~pile_entier()
{
   delete pile; // libère la mémoire
}

 void pile_entier::empile(int p)
{
   *(pile+hauteur) = p; hauteur++;
}

 int pile_entier::depile()
{
   hauteur--;
   int res = *(pile+hauteur);
   return res;
}

 int pile_entier::pleine()
{
   if(hauteur==taille)
      return 1;
   return 0;
}

 int pile_entier::vide()
{
   if(hauteur==0)
      return 1;
   return 0;
}

 void main()
{
   pile_entier a, b(15);   // pile automatique
   a.empile(8);
   if(a.vide()==1) cout<<"a vide\n";
   else cout<<"a non vide\n";

   pile_entier *adp;          // pile dynamique
   adp = new pile_entier(5);  // pointeur sur pile de 5 entiers
   for(int i=0;adp->pleine()!=1;i++) adp->empile(10*i);
   cout<<"\nContenu de la pile dynamique:\n";
   for(int i=0;i<5;i++)
      if(adp->vide()!=1) cout<<adp->depile()<<"\n";
   getch();
}
```

Exercice XII-14:

```
#include <iostream.h>  // constructeur par recopie
#include <conio.h>

    class  pile_entier
   {
   private :
      int *pile,taille,hauteur;
   public:
      pile_entier(int n); // constructeur, taille de la pile
      pile_entier(pile_entier &p);  // constructeur par recopie
      ~pile_entier(); // destructeur
      void empile(int p); // ajoute un élément
      int depile();       // dépile un élément
      int pleine();       // 1 si vrai   0 sinon
      int vide();         // 1 si vrai  0 sinon
   };

    pile_entier::pile_entier(int n=20)  // taille par défaut: 20
   {
      taille = n;
      pile = new int[taille];  // taille de la pile
      hauteur = 0;
      cout<<"On a fabrique une pile de "<<taille<<" éléments\n";
      cout<<"Adresse de la pile: "<<pile<<" ; de l'objet: "<<this<<"\n";
   }

    pile_entier::pile_entier(pile_entier &p)
   {
      taille = p.taille; hauteur = p.hauteur;
      pile=new int[taille];
      for(int i=0;i<hauteur;i++)*(pile+i) = p.pile[i];
      cout<<"On a fabriqué une pile de "<<taille<<" éléments\n";
      cout<<"Adresse de la pile: "<<pile<<" ; de l'objet: "<<this<<"\n";
   }

    pile_entier::~pile_entier()
   {
      delete pile; // libère la mémoire
   }

    void pile_entier::empile(int p)
   {
      *(pile+hauteur) = p; hauteur++;
   }

    int pile_entier::depile()
   {
      hauteur--;
      int res=*(pile+hauteur);
      return res;
   }
```

```
 int pile_entier::pleine()
{
   if(hauteur==taille)
      return 1;
   else
      return 0;
}

 int pile_entier::vide()
{
   if(hauteur==0)
      return 1;
   else
      return 0;
}

 void main()
{
   cout<<"Pile a:\n";pile_entier a(10);
   for(int i=0;a.pleine()!=1;i++)a.empile(2*i);
   cout<<"Pile b:\n";
   pile_entier b = a;
   while(b.vide()!=1)cout<<b.depile()<<"  "; getch();
}
```

Exercice XII-15:

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

// Tableau de vecteurs

 class vecteur
{
private:
   float x,y;
public: vecteur(float abs,float ord);
   void  homothetie(float val);
   void affiche();
   float det(vecteur w);
};

 vecteur::vecteur(float abs =5.0,float ord = 3.0)
{
   x = abs;
   y = ord;
}

 void vecteur::homothetie(float val)
{
   x = x*val;
   y = y*val;
}
```

```
 void vecteur::affiche()
{
   cout<<"x = "<< x <<"  y = "<< y <<"\n";
}

 float vecteur::det(vecteur w)
{
   float res = x * w.y - y * w.x;
   return res;
}

 void main()
{
   vecteur v[4]={17,9},*u;
   u = new vecteur[3];  // tableau de 3 vecteurs
   for(int i=0;i<4;i++)
   {
      v[i].affiche();
   }
   v[2].homothetie(3);
   v[2].affiche();

   cout <<"Determinant de (u1,v0) = "<<v[0].det(u[1])<<"\n";
   cout <<"Determinant de (v2,u2) = "<<u[2].det(v[2])<<"\n";

   delete []u; // noter les crochets

   getch();
}
```

COURS et TP DE LANGAGE C++

Chapitre 13

Surcharge des opérateurs

Joëlle MAILLEFERT

joelle.maillefert@iut-cachan.u-psud.fr

IUT de CACHAN

Département GEII 2

CHAPITRE 13

SURCHARGE DES OPERATEURS

I- INTRODUCTION

Le langage C++ autorise le programmeur à étendre la signification d'opérateurs tels que l'addition (+), la soustraction (-), la multiplication (*), la division (/), le ET logique (&) etc...

Exemple:

On reprend la classe vecteur déjà étudiée et on surdéfinit l'opérateur somme (+) qui permettra d'écrire dans un programme:

vecteur v1, v2, v3;
v3 = v2 + v1;

Exercice XIII-1:

Etudier et tester le programme suivant:

```
#include <iostream.h>
#include <conio.h>
// Classe vecteur : surcharge de l'opérateur +

    class vecteur
   {
   private:
      float x,y;
   public:
      vecteur(float abs,float ord);
      void affiche();
    // surcharge de l'opérateur somme, on passe un paramètre vecteur
    // la fonction retourne un vecteur
      vecteur operator + (vecteur v);
   };

    vecteur::vecteur(float abs =0,float ord = 0)
   {
      x = abs; y = ord;
   }

    void vecteur::affiche()
   {
      cout<<"x = "<< x <<"  y = "<< y <<"\n";
   }
```

```
 vecteur vecteur::operator+(vecteur v)
{
   vecteur res;
   res.x = v.x + x; res.y = v.y + y;
   return res;
}

  void main()
 {
    vecteur a(2,6),b(4,8),c,d,e,f;
    c = a + b;            c.affiche();
    d = a.operator+(b);   d.affiche();
    e = b.operator+(a);   e.affiche();
    f = a + b + c;        f.affiche();
    getch() ;
 }
```

Exercice XIII-2:

Ajouter une fonction membre de prototype **float operator*(vecteur v)** permettant de créer l'opérateur « produit scalaire », c'est à dire de donner une signification à l'opération suivante:

vecteur v1, v2;
float prod_scal;
prod_scal = v1 * v2;

Exercice XIII-3:

Ajouter une fonction membre de prototype **vecteur operator*(float)** permettant de donner une signification au produit d'un réel et d'un vecteur selon le modèle suivant :

vecteur v1,v2;
float h;
v2 = v1 * h ; // homothétie

Les arguments étant de type différent, cette fonction peut cohabiter avec la précédente.

Exercice XIII-4:

Sans modifier la fonction précédente, essayer l'opération suivante et conclure.

vecteur v1,v2;
float h;
v2 = h * v1; // homothétie

Cette appel conduit à une erreur de compilation. L'opérateur ainsi créé, n'est donc pas symétrique. Il faudrait disposer de la notion de « fonction amie » pour le rendre symétrique.

II- APPLICATION: UTILISATION D'UNE BIBLIOTHEQUE

BORLAND C++ possède une classe « complex », dont le prototype est déclaré dans le fichier **complex.h**.

Voici une partie de ce prototype:

```
 class complex
{
private:
   double re,im; // partie réelle et imaginaire d'un nombre complexe
public:
   complex(double reel, double imaginaire = 0); // constructeur
 // complex manipulations
   double real(complex);   // retourne la partie réelle
   double imag(complex);   // retourne la partie imaginaire
   complex conj(complex);  // the complex conjugate
   double norm(complex);   // the square of the magnitude
   double arg(complex);    // the angle in radians
 // Create a complex object given polar coordinates
   complex polar(double mag, double angle=0);
 // Binary Operator Functions
   complex  operator+(complex);
 // donnent un sens à : « complex + double » et « double + complex »
   friend complex  operator+(double, complex);
   friend complex  operator+(complex, double);
 // la notion de « fonction amie » sera étudiée au prochain chapitre
   complex  operator-(complex);
   friend complex  operator-(double, complex);
 // idem avec la soustraction
   friend complex  operator-(complex, double);
   complex  operator*(complex);
   friend complex  operator*(complex, double);
 // idem avec la multiplication
   friend complex  operator*(double, complex);
   complex  operator/(complex);
   friend complex  operator/(complex, double);
 // idem avec la division
   friend complex  operator/(double, complex);
   int  operator==(complex);          // retourne 1 si égalité
   int  operator!=(complex, complex); // retourne 1 si non égalité
   complex  operator-();              // opposé du vecteur
};
```

```
// Complex stream I/O
// permet d'utiliser cout avec un complexe
   ostream    operator<<(ostream  , complex);
// permet d'utiliser cin avec un complexe
   istream    operator>>(istream  , complex);
```

Exercice XIII-5:
Analyser le fichier **complex.h** pour identifier toutes les manipulations possibles avec les nombres complexes en BORLAND C++.
Ecrire une application.

III- REMARQUES GENERALES (***)

- Pratiquement tous les opérateurs peuvent être surdéfinis:
+ - * / = ++ -- new delete [] -> & | ^ && || % << >> etc ...
avec parfois des règles particulières non étudiées ici.
- Il faut se limiter aux opérateurs existants.
- Les règles d'associativité et de priorité sont maintenues.
- Il n'en est pas de même pour la commutativité (cf exercice XIII-3 et XIII-4).
- L'opérateur = peut-être redéfini. S'il ne l'est pas, une recopie est exécutée comme on l'a vu dans le chapitre II (cf exercice II-1, à re-tester).
Si la classe contient des données dynamiques, il faut impérativement surcharger l'opérateur =.

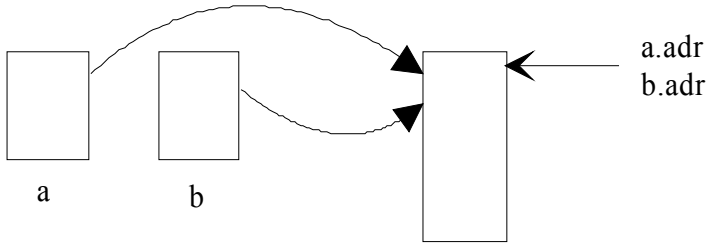
Exercice XIII-6:

Dans le programme ci-dessous, on surdéfinit l'opérateur =.
En étudier soigneusement la syntaxe, tester avec et sans la surcharge de l'opérateur =.
Conclure.

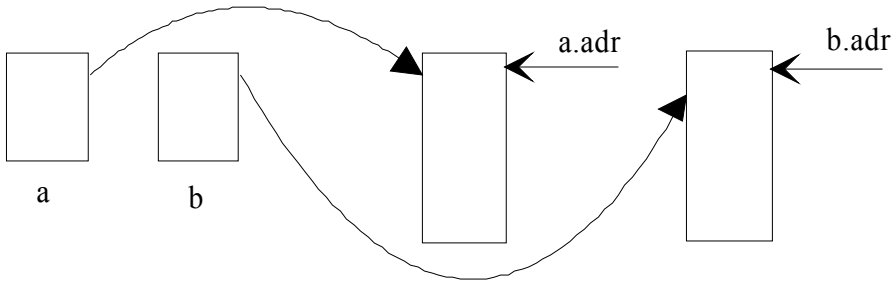
```
#include <iostream.h>
#include <conio.h>

    class liste
   {
   private:
      int taille;
      float *adr;
   public:
      liste(int t);            // constructeur
      liste::liste(liste &v);  // constructeur par recopie
      void saisie(); void affiche();
      void operator=(liste &); // surcharge de l'opérateur =
      ~liste();
   };
```

```
 liste::liste(int t)
{
   taille = t;adr = new float[taille];cout<<"Construction";
   cout<<" Adresse de l'objet:"<< this;
   cout<<" Adresse de liste:"<< adr <<"\n";
}

 liste::~liste()
{
   cout<<"Destruction Adresse de l'objet:"<< this;
   cout<<" Adresse de liste:"<< adr <<"\n"; delete adr;
}

 liste::liste(liste &v)
{
   taille = v.taille;adr = new float[taille];
   for(int i=0;i<taille;i++)adr[i] = v.adr[i];
   cout<<"\nConstructeur par recopie";
   cout<<"  Adresse de l'objet:"<< this;
   cout<<"  Adresse de liste:"<< adr <<"\n";}

 void liste::saisie()
{
   for(int i=0;i<taille;i++)
   {cout<<"Entrer un nombre:";cin>>*(adr+i);}
}

 void liste::affiche()
{
   cout<<"Adresse:"<<this<<" ";
   for(int i=0;i<taille;i++)cout<<*(adr+i)<<" ";
   cout<<"\n\n";
}

 void liste::operator=(liste &lis)
{/* passage par référence pour éviter l'appel au constructeur par
 * recopie et la double libération d'un même emplacement mémoire */
   taille=lis.taille; delete adr; adr=new float[taille];
   for(int i=0;i<taille;i++)adr[i] = lis.adr[i];
}

 void main()
{
   cout<<"Debut de main()\n";
   liste a(5); liste b(2);
   a.saisie();  a.affiche();
   b.saisie();  b.affiche();
   b = a;
   b.affiche(); a.affiche();
   cout<<"Fin de main()\n";
}
```

On constate donc que la surcharge de l'opérateur = permet d'éviter la situation suivante:



et conduit à:



Conclusion:

Une classe qui présente une donnée membre allouée dynamiquement doit toujours posséder au minimum, un constructeur, un constructeur par recopie, un destructeur, la surcharge de l'opérateur =. Une classe qui possède ces propriétés est appelée « classe canonique ».

IV- EXERCICES RECAPITULATIFS (***)

Exercice XIII-7:

Reprendre la classe pile_entier de l'exercice IV-13 et remplacer la fonction membre « empile » par l'opérateur < et la fonction membre « depile » par l'opérateur >.
p < n ajoute la valeur n sur la pile p
p> n supprime la valeur du haut de la pile p et la place dans n.

Exercice XIII-8:

Ajouter à cette classe un constructeur par recopie et la surdéfinition de l'operateur =

Exercice XIII-9:

Ajouter à la classe **liste** la surdéfinition de l'opérateur [], de sorte que la notation a[i] ait un sens et retourne l'élément d'emplacement i de la liste a.
Utiliser ce nouvel opérateur dans les fonctions **affiche** et **saisie**
On créera donc une fonction membre de prototype ***float &liste::operator[](int i);***

Exercice XIII-10:

Définir une classe **chaine** permettant de créer et de manipuler une chaîne de caractères:
données:

- longueur de la chaine (entier)
- adresse d'une zône allouée dynamiquement (inutile d'y ranger la constante \0)

méthodes:

- constructeur **chaine()** initialise une chaîne vide
- constructeur **chaine(char *texte)** initialise avec la chaîne passée en argument
- constructeur par recopie **chaine(chaine &ch)**
- opérateurs affectation (=), comparaison (==), concaténation (+), accès à un caractère de rang donné ([])

V- CORRIGE DES EXERCICES

Exercice XIII-2:

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

// Classe vecteur, surcharge de l'opérateur produit scalaire

    class vecteur
   {
   private:
      float x,y;
   public: vecteur(float abs,float ord);
      void affiche();
      vecteur operator+(vecteur v); // surcharge de l'operateur +
    // surcharge de l'opérateur * en produit scalaire
      float operator*(vecteur v);
   };

    vecteur::vecteur(float abs =0,float ord = 0)
   {
      x = abs; y = ord;
   }

    void vecteur::affiche()
   {cout<<"x = "<<x<<"  y = "<<y<<"\n";}

    vecteur vecteur::operator+(vecteur v)
   {
      vecteur res; res.x = v.x + x; res.y = v.y + y;
      return res;
   }

    float vecteur::operator*(vecteur v)
   {
      float res = v.x * x + v.y * y;
      return res;
   }

    void main()
   {
      vecteur a(2,6),b(4,8),c;
      float prdscl1,prdscl2,prdscl3;
      c = a + b; c.affiche();
      prdscl1 = a * b;
      prdscl2 = a.operator*(b);
      prdscl3 = b.operator*(a);
      cout<<prdscl1<<" "<<prdscl2<<" "<<prdscl3<<"\n"; getch();
   }
```

Exercice XIII-3:

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

    class vecteur
   {
   private:
      float x,y;
   public: vecteur(float abs,float ord);
      void affiche();
      vecteur operator+(vecteur v); // surcharge de l'opérateur +
      float operator*(vecteur v);   // surcharge de l'opérateur *
                                    // produit scalaire
      vecteur operator*(float f);   // surcharge de l'opérateur *
                                    // homothétie
   };

    vecteur::vecteur(float abs =0,float ord = 0)
   {
      x = abs; y = ord;
   }

    void vecteur::affiche()
   {cout<<"x = "<< x <<"  y = "<< y <<"\n";}

    vecteur vecteur::operator+(vecteur v)
   {
      vecteur res; res.x = v.x + x; res.y = v.y + y;
      return res;
   }

    float vecteur::operator*(vecteur v)
   {
      return v.x * x + v.y * y;
   }

    vecteur vecteur::operator*(float f)
   {
      vecteur res; res.x = f*x;   res.y = f*y;
      return res;
   }

    void main()
   {
      vecteur a(2,6),b(4,8),c,d;
      float prdscl1,h=2.0;
      c = a + b; c.affiche();
      prdscl1 = a * b; cout<<prdscl1<<"\n";
      d =  a * h; d.affiche();getch() ;
   }
```

Exercice XIII-5:

```
#include <iostream.h>  // Utilisation de la bibliothèque
#include <conio.h>     // de manipulation des nombres complexes
#include <complex.h>

#define PI 3.14159

    void main()
   {
      complex a(6,6),b(4,8),c(5);
      float n =2.0,x,y;
      cout<<"a = "<< a <<" b= "<< b <<" c= "<< c <<"\n" ;
      c = a + b;              cout<<"c= "<< c <<"\n" ;
      c = a * b;              cout<<"c= "<< c <<"\n" ;
      c = n * a;              cout<<"c= "<< c <<"\n" ;
      c = a * n;              cout<<"c= "<< c <<"\n" ;
      c = a/b;                cout<<"c= "<< c <<"\n" ;
      c = a/n;                cout<<"c= "<< c <<"\n" ;
      x = norm(a); cout<<"x= "<< x <<"\n" ;
      y = arg(a)*180/PI;  // Pour l'avoir en degrés
      cout<<"y= "<< y <<"\n" ;
      c = polar(20,PI/6); // module = 20  angle = 30°
      cout<<"c= "<< c <<"\n" ;
      c = -a;   cout<<"c= "<< c <<"\n" ;
      c = a+n;  cout<<"c= "<< c <<"\n" ;
      cout<< (c==a) <<"\n" ;
      cout<<"Saisir c sous la forme (re,im): ";
      cin >> c;
      cout<<"c= "<< c <<"\n" ;
      getch();
   }
```

Exercice XIII-7:

```
    class  pile_entier
   {
   private :
      int *pile,taille,hauteur;
   public:
      pile_entier(int n); // constructeur, taille de la pile
      ~pile_entier();     // destructeur
      void operator < (int x);  // ajoute un élément
      void operator >(int &x);  // dépile un élément
      int pleine();             // 1 si vrai 0 sinon
      int vide();               // 1 si vrai 0 sinon
   };
```

```
 pile_entier::pile_entier(int n=20) // taille par défaut: 20
{
   taille = n;
   pile = new int[taille];  // taille de la pile
   hauteur = 0;
   cout<<"On a fabriqué une pile de "<< taille <<" éléments\n";
}

 pile_entier::~pile_entier()
{
   delete pile;// libère la mémoire
}

 void pile_entier::operator<(int x)
{
   *(pile+hauteur) = x; hauteur++;
}

 void pile_entier::operator >(int &x)
{ // passage par référence obligatoire (modification de l'argument)
   hauteur--; x = *(pile+hauteur);
}

 int pile_entier::pleine()
{
   if(hauteur==taille)
      return 1;
   return 0;
}

 int pile_entier::vide()
{
   if(hauteur==0)
      return 1;
   return 0;
}


 void main()
{
   pile_entier a ;
   int n = 8,m;
   a < n;
   if (a.vide()) cout<<"La pile est vide\n";
   else cout<<"La pile n'est pas vide\n";
   a > m;
   cout<<"m="<< m <<"\n";
   if (a.vide()) cout<<"La pile est vide\n";
   else cout<<"La pile n'est pas vide\n";
   getch();
}
```

Exercice XIII-8: On ajoute les éléments suivants:

```
#include <iostream.h> // Gestion d'une pile d'entiers
#include <conio.h>

    class  pile_entier
   {
   private :
      int *pile,taille,hauteur;
   public:
      pile_entier(int n); // constructeur, taille de la pile
      pile_entier(pile_entier &p); // constructeur par recopie
      ~pile_entier(); // destructeur
      void operator < (int x);  // ajoute un élément
      void operator >(int &x);  // dépile un élément
      void operator = (pile_entier &p); // surcharge de l'operateur =
      int pleine();             // 1 si vrai 0 sinon
      int vide();               // 1 si vrai 0 sinon
   };

    pile_entier::pile_entier(int n=20)  // taille par défaut: 20
   {
      taille = n; // taille de la pile
      pile = new int[taille]; hauteur = 0;
      cout<<"On a fabriqué une pile de "<<taille<<" éléments\n";
   }

    pile_entier::pile_entier(pile_entier &p) // constructeur par recopie
   {
      taille = p.taille;
      pile =  new int [taille] ; hauteur =  p.hauteur;
      for(int i=0;i<hauteur;i++)*(pile+i) = p.pile[i];
      cout<<"On a bien recopié la pile\n";
   }

    pile_entier::~pile_entier()
   {
      delete pile; // libère la mémoire
   }

    void pile_entier::operator<(int x)
   {
      *(pile+hauteur) = x; hauteur++;
   }

    void pile_entier::operator >(int &x)
   {// passage par référence obligatoire (modifier valeur de l'argument)
      hauteur--; x = *(pile+hauteur);
   }
```

```
 int pile_entier::pleine()
{
   if(hauteur==taille)
      return 1;
   return 0;
}

 int pile_entier::vide()
{
   if(hauteur==0)
      return 1;
   return 0;
}

 void pile_entier::operator = (pile_entier &p)
{
   taille = p.taille;
   pile = new int [taille];
   hauteur = p.hauteur;
   for(int i=0;i<hauteur;i++)
      *(pile+i)=p.pile[i];
   cout<<"l'égalité, ça marche !\n";
}

 void main()
{
   pile_entier a,c(10);
   int m,r,s ;
   for(int n=5;n<22;n++)
      a < n;            // empile 18 valeurs
   pile_entier b = a;
   for(int i=0; i<3;i++)
   {
      b>m;cout<<m<<" "; // dépile  3 valeurs
   }
   cout<<"\n";c = a;
   for(int i=0;i<13;i++)
   {
      c>r;cout<<r<<" "; // dépile 13 valeurs
   }
   cout<<"\n";
   for(int i=0; i<4;i++)
   {
      a>s;cout<<s<<" "; // dépile  4 valeurs
   }
   getch();
}
```

Exercice XIII-9:

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

    class liste   // NOTER LA MODIFICATION DES FONCTIONS
    // Fonctions saisie ET affiche QUI UTILISENT L'OPERATEUR []
   {
   private:
      int taille;
      float *adr;
   public:
      liste(int t);
      liste(liste &v);
      void operator=(liste &lis); // surcharge de l'opérateur =
      float &operator[](int i); // surcharge de l'opérateur []
      void saisie();
      void affiche();
      ~liste();
   };

    liste::liste(int t)
   {
      taille = t;adr = new float[taille];cout<<"Construction";
      cout<<" Adresse de l'objet:"<< this;
      cout<<" Adresse de liste:"<< adr <<"\n";
   }

    liste::~liste()
   {
      cout<<"Destruction Adresse de l'objet : "<< this;
      cout<<" Adresse de liste : "<< adr <<"\n";
      delete adr;
   }

    liste::liste(liste &v)
   {
      taille = v.taille; adr = new float[taille];
      for(int i=0;i<taille;i++)adr[i] = v.adr[i];
      cout<<"\nConstructeur par recopie";
      cout<<"  Adresse de l'objet : "<< this;
      cout<<"  Adresse de liste : "<< adr <<"\n";
   }

    void liste::operator=(liste &lis)
   { /* passage par référence pour éviter l'appel au constructeur par
     * recopie et la double libération d'un même emplacement mémoire */
      taille=lis.taille;
      delete adr; adr=new float[taille];
      for(int i=0;i<taille;i++) adr[i] = lis.adr[i];
   }
```

```
 float &liste::operator[](int i) // surcharge de []
{
   return adr[i];
}

 void liste::saisie()  // UTILISATION DE []
{
   for(int i=0;i<taille;i++)
   {cout<<"Entrer un nombre:";cin>>adr[i];}
}

 void liste::affiche() // UTILISATION DE []
{
   cout<<"Adresse:"<< this <<" ";
   for(int i=0;i<taille;i++) cout<<adr[i]<<" ";
   cout<<"\n\n";
}

 void main()
{
   cout<<"Début de main()\n";
   liste a(3);
   a[0]=25; a[1]=233;
   cout<<"Saisir un nombre:";
   cin>>a[2];  a.affiche();
   a.saisie(); a.affiche();
   cout<<"Fin de main()\n";
   getch();
}
```

Exercice XIII-10:

```
#include <iostream.h> // classe chaine
#include <conio.h>

 class chaine
{
private :
   int longueur; char *adr;
public:
   chaine();chaine(char *texte);chaine(chaine &ch);  //constructeurs
   ~chaine();
   void operator=(chaine &ch);
   int operator==(chaine ch);
   chaine &operator+(chaine ch);
   char &operator[](int i);
   void affiche();
};
```

```
 chaine::chaine(){longueur = 0;adr = new char[1];} //constructeur1

 chaine::chaine(char *texte) // constructeur2
{
   for(int i=0;texte[i]!='\0';i++);
   longueur = i;
   adr = new char[longueur+1];
   for(int i=0;i!=(longueur+1);i++) adr[i] = texte[i];
}

 chaine::chaine(chaine &ch) //constructeur par recopie
{
   longueur = ch.longueur;
   adr = new char[longueur];
   for(int i=0;i!=(longueur+1);i++)adr[i] = ch.adr[i];
}

 void chaine::operator=(chaine &ch)
{
   delete adr;
   longueur = ch.longueur;
   adr = new char[ch.longueur+1];
   for(int i=0;i!=(longueur+1);i++) adr[i] = ch.adr[i];
}

 int chaine::operator==(chaine ch)
{
   res=1;
   for(int i=0;(i!=(longueur+1))&&(res!=0);i++)
      if(adr[i]!=ch.adr[i])res=0;
   return res;
}

 chaine &chaine::operator+(chaine ch)
{
   static chaine res;
   res.longueur = longueur + ch.longueur;
   res.adr = new char[res.longueur+1];
   for(int i=0;i!=longueur;i++) res.adr[i] = adr[i];
   for(int i=0;i!=ch.longueur;i++)res.adr[i+longueur] = ch.adr[i];
   res.adr[res.longueur]='\0';
   return(res);
}

 char &chaine::operator[](int i)
{
   static char res='\0';
   if(longueur!=0) res =  *(adr+i);
   return res;
}
```

```
 chaine::~chaine()
{
   delete adr;
}

 void chaine::affiche()
{
   for(int i=0;i!=longueur;i++)
   {
      cout<<adr[i];
   }
   cout<<"\n";
}

 void main()
{
   chaine a("Bonjour "),b("Maria"),c,d("Bonjour "),e;
   if(a==b)cout<<"Gagné !\n";
   else    cout<<"Perdu !\n";
   if(a==d)cout<<"Gagné !\n";
   else    cout<<"Perdu !\n";
   cout<<"a: ";  a.affiche();
   cout<<"b: ";  b.affiche();
   cout<<"d: ";  d.affiche();

   c = a+b;
   cout<<"c: ";  c.affiche();

   for(int i=0;c[i]!='\0';i++) cout<<c[i];
   getch();
}
```

COURS et TP DE LANGAGE C++

Chapitre 14

Fonctions amies

Joëlle MAILLEFERT

joelle.maillefert@iut-cachan.u-psud.fr

IUT de CACHAN

Département GEII 2

CHAPITRE 14

FONCTIONS AMIES

Grâce aux fonctions amies, on pourra accéder aux membres privés d'une classe, autrement que par le biais de ses fonctions membres publiques.

Il existe plusieurs situations d'amitié:

- Une fonction indépendante est amie d'une ou de plusieurs classes.
- Une ou plusieurs fonctions membres d'une classe sont amie d'une autre classe.

I- FONCTION INDEPENDANTE AMIE D'UNE CLASSE

Exemple (à tester) et exercice XIV-1:

Dans l'exemple ci-dessous, la fonction *coïncide* est AMIE de la classe *point.* C'est une fonction ordinaire qui peut manipuler les membres privés de la classe *point*.

```
#include <iostream.h> // fonction indépendante, amie d'une classe
#include <conio.h>

    class point
   {
   private :
      int x,y;
   public:
       point(int abs=0,int ord=0){x=abs;y=ord;}
   //déclaration de la fonction amie
      friend int coincide(point,point);
   };

    int coincide(point p,point q)
   {
      if((p.x==q.x)&&(p.y==q.y))
         return 1;
      return 0;
   }

    void main()
   {point a(4,0),b(4),c;
      if(coincide(a,b))cout<<"a coïncide avec b\n";
      else cout<<"a est différent de b\n";
      if(coincide(a,c))cout<<"a coïncide avec c\n";
      else cout<<"a est différent de c\n";
      getch() ;
   }
```

Exercice XIV-2:

Reprendre l'exercice III-8 dans lequel une fonction membre de la classe **vecteur** permettait de calculer le déterminant de deux vecteurs:
Définir cette fois-ci une fonction indépendante AMIE de la classe vecteur.

II- LES AUTRES SITUATIONS D'AMITIE

1- Dans la situation ci-dessous, la fonction **fm_de_titi**, fonction membre de la classe TITI, a accès aux membres privés de la classe TOTO:

```
class TOTO
{
// partie privée
.....
……
// partie publique
friend int TITI::fm_de_titi(char, TOTO);
};

class TITI
{
.....
……
int fm_de_titi(char, TOTO);
};

int TITI::fm_de_titi(char c, TOTO t)
{
...
} // on pourra trouver ici une invocation des membres privés de l'objet t
```

Si toutes les fonctions membres de la classe TITI étaient amies de la classe TOTO, on déclarerait directement dans la partie publique de la classe TOTO: **friend class TITI;**

2- Dans la situation ci-dessous, la fonction **f_anonyme** a accès aux membres privés des classes TOTO et TITI:

```
class TOTO
{
// partie privée
.....
.....
// partie publique
friend void f_anonyme(TOTO, TITI);
};
```

```
class TITI
{
// partie privée
........
........
// partie publique
friend void f_anonyme(TOTO, TITI);
};

void f_anonyme(TOTO to, TITI ti)
{
...
} // on pourra trouver ici une invocation des membres privés des objets to et ti.
```

III- APPLICATION A LA SURCHARGE DES OPERATEURS

Exemple (à tester) et exercice XIV-3:

On reprend l'exemple V-1 permettant de surcharger l'opérateur + pour l'addition de 2 vecteurs.
On crée, cette fois-ci, une fonction AMIE de la classe **vecteur.**

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

// Classe vecteur
// Surcharge de l'opérateur + par une fonction AMIE

    class vecteur
   {
   private:
      float x,y;
   public: vecteur(float,float);
      void affiche();
      friend vecteur operator+(vecteur, vecteur);
   };

    vecteur::vecteur(float abs =0,float ord = 0)
   {x=abs; y=ord;}

    void vecteur::affiche()
   {cout<<"x = "<< x << " y = " << y << "\n";}

    vecteur operator+(vecteur v, vecteur w)
   {
      vecteur res;
      res.x = v.x + w.x;
      res.y = v.y + w.y;
      return res;
   }

    void main()
   {vecteur a(2,6),b(4,8),c,d;
      c = a + b; c.affiche();
      d = a + b + c; d.affiche();
      getch() ;
   }
```

Exercice XIV-4:

Reprendre l'exercice XIV-1: redéfinir l'opérateur == correspondant à la fonction **coïncide.**

Exercice XIV-5:

Reprendre les exercices V-2, V-3 et V-4: En utilisant la propriété de surcharge des fonctions du C++, créer
- une fonction membre de la classe **vecteur** de prototype
float vecteur::operator*(vecteur); qui retourne le produit scalaire de 2 vecteurs
- une fonction membre de la classe **vecteur** de prototype
vecteur vecteur::operator*(float); qui retourne le vecteur produit d'un vecteur et d'un réel (donne une signification à **v2 = v1 * h;**)
- une fonction AMIE de la classe **vecteur** de prototype
vecteur operator*(float, vecteur); qui retourne le vecteur produit d'un réel et d'un vecteur (donne une signification à **v2 = h * v1;**)

On doit donc pouvoir écrire dans le programme:

```
vecteur v1, v2, v3, v4;
float h, p;
p = v1 * v2;
v3 = h * v1;
v4 = v1 * h;
```

Remarque:
On aurait pu remplacer la fonction membre de prototype **vecteur vecteur::operator*(float);** par une fonction AMIE de prototype **vecteur operator*(vecteur, float);**

Exercice XIV-6:

Etudier le listing du fichier d'en-tête **complex.h** fourni au chapitre V et justifier tous les prototypes des fonctions.

IV- CORRIGE DES EXERCICES

Exercice XIV-2:

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

 // Classe vecteur
 // Fonction AMIE calculant le déterminant de 2 vecteurs

    class vecteur
   {
   private :
      float x,y;
   public: vecteur(float,float);
      void affiche();
      friend float det(vecteur, vecteur);
   };

    vecteur::vecteur(float abs =0.,float ord = 0.)
   {x=abs; y=ord;}

    void vecteur::affiche()
   {cout<<"x = "<< x << " y = "<< y << "\n";}

    float det(vecteur a, vecteur b)
   { // la fonction AMIE peut manipuler
     // les quantités b.x, b.y, a.x, a.y
      float res;
      res = a.x * b.y - a.y * b.x;
      return res;
   }

    void main()
   {
      vecteur u(2,6),v(4,8);
      u.affiche(); v.affiche();
      cout <<"Déterminant de (u,v) = "<<det(u,v)<<"\n";
      cout <<"Déterminant de (v,u) = "<<det(v,u)<<"\n";
      getch() ;
   }
```

Exercice XIV-4:

```
#include <iostream.h>// Surcharge de l'opérateur ==

    class point
   {
   private:
      int x,y;
   public:
       point(int abs=0,int ord=0)
      {
         x=abs; y=ord;
      }
   // déclaration de la fonction amie
      friend int operator==(point,point);
   };

    int operator ==(point p, point q)
   {
      if((p.x==q.x)&&(p.y==q.y))
         return 1;
      return 0;
   }

    void main()
   {
      point a(4,0),b(4),c;
      if(a==b)cout<<"a coïncide avec b\n";
      else cout<<"a est différent de b\n";
      if(a==c)cout<<"a coïncide avec c\n";
      else cout<<"a est différent de c\n";
      getch();
   }
```

Exercice XIV-5:

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

    class vecteur
   {
   private:
      float x,y;
   public:
      vecteur(float,float);
      void affiche(); // surcharger +
      vecteur operator+(vecteur); // surcharger * : produit scalaire
      float operator*(vecteur);
   // surcharger * : vecteur (passé en paramètre)*scalaire retourné
      vecteur operator*(float);
   // surcharger * : vecteur (objet)*scalaire retourné
      friend vecteur operator*(float,vecteur);
   };

    vecteur::vecteur(float abs =0,float ord = 0)
   {x=abs;y=ord;}
    void vecteur::affiche()
   {cout<<"x = "<<x<<"  y = "<<y<<"\n";}
    vecteur vecteur::operator+(vecteur v)
   {
      vecteur res; res.x = v.x + x; res.y = v.y + y;
      return res;
   }
    float vecteur::operator*(vecteur v)
   {
      float res = v.x * x + v.y * y;
      return res;
   }
    vecteur vecteur::operator*(float f)
   {
      vecteur res; res.x = f*x; res.y = f*y;
      return res;
   }
    vecteur operator*(float f, vecteur v)
   {
      vecteur res; res.x = f*v.x; res.y = f*v.y;
      return res;
   }

    void main()
   {
      vecteur a(2,6),b(4,8),c,d; float p,h=2.0;
      p = a * b; cout<< p <<"\n";
      c = h * a; c.affiche(); d = a * h; d.affiche(); getch();
   }
```

COURS et TP DE LANGAGE C++

Chapitre 15

L'héritage

Joëlle MAILLEFERT

joelle.maillefert@iut-cachan.u-psud.fr

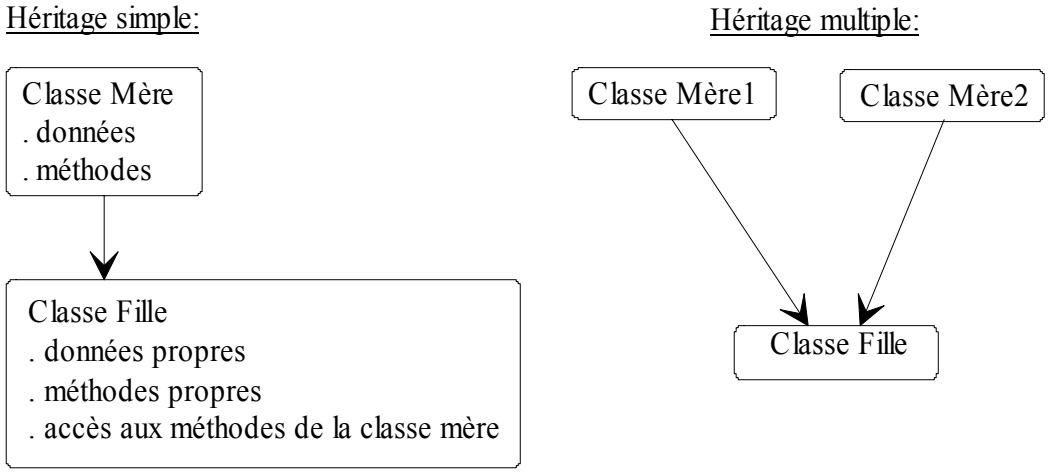
IUT de CACHAN

Département GEII 2

CHAPITRE 15

L'HERITAGE

La P.O.O. permet de définir de nouvelles classes (classes filles) dérivées de classes de base (classes mères), avec de nouvelles potentialités. Ceci permettra à l'utilisateur, à partir d'une bibliothèque de classes donnée, de développer ses propres classes munies de fonctionnalités propres à l'application.
On dit qu'une classe fille DERIVE d'une ou de plusieurs classes mères.



La classe fille n'a pas accès aux données (privées) de la classe mère.

I- DERIVATION DES FONCTIONS MEMBRES

Exemple (à tester) et exercice XV-1:
L'exemple ci-dessous illustre les mécanismes de base :

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

    class vecteur  // classe mère
   {

   private:
      float x,y;
   public:
      void initialise(float,float);
      void  homothetie(float);
      void affiche();
   };
```

```
 void vecteur::initialise(float abs =0.,float ord = 0.)
{
   x=abs; y=ord;
}

 void vecteur:: homothetie(float val)
{
   x = x*val; y = y*val;
}
 void vecteur::affiche()
{
   cout<<"x = "<<x<<"  y = "<<y<<"\n";
}
 // classe fille
 class vecteur3:public vecteur
{
private :
   float z;
public:
   void initialise3(float,float,float);
   void  homothetie3(float);
    void hauteur(float ha){z = ha;}
   void affiche3();
};

 void vecteur3::initialise3(float abs=0.,float ord=0.,float haut=0.)
{  // appel de la fonction membre de la classe vecteur
   initialise(abs,ord); z = haut;
}

 void vecteur3:: homothetie3(float val)
{  // appel de la fonction membre de la classe vecteur
   homothetie(val);  z = z*val;
}

 void vecteur3::affiche3()
{  // appel de la fonction membre de la classe vecteur
   affiche();cout<<"z = "<<z<<"\n";
}

 void main()
{
   vecteur3 v, w;
   v.initialise3(5,4,3);v.affiche3();  // fonctions de la fille
   w.initialise(8,2);w.hauteur(7);w.affiche(); // fonctions de la mère
   cout<<"*******************************\n";
   w.affiche3();w.homothetie3(6);w.affiche3(); //fonctions de la fille
   getch();
}
```

L'exemple ci-dessus présente une syntaxe assez lourde et très dangereuse. Il serait plus simple, pour l'utilisateur, de donner aux fonctions membres de la classe fille, le même nom que dans la classe mère, lorsque celles-ci jouent le même rôle (ici fonctions **initialise** et **homothétie**).
Ceci est possible en utilisant la propriété de ***redéfinition*** des fonctions membres.

Exemple (à tester) et exercice XV-2:

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

    class vecteur  // classe mère
   {
   private:
      float x,y;
   public: void initialise(float,float);
      void  homothetie(float);
      void affiche();
   };

    void vecteur::initialise(float abs =0.,float ord = 0.)
   {
      x=abs; y=ord;
   }
    void vecteur::homothetie(float val)
   {
      x = x*val; y = y*val;
   }

    void vecteur::affiche()
   {
      cout<<"x = "<<x<<"  y = "<<y<<"\n";
   }

    class vecteur3:public vecteur // classe fille
   {
   private :
      float z;
   public:
      void initialise(float,float,float);
      void  homothetie(float);
       void hauteur(float ha)
      {
         z = ha;
      }
      void affiche();
   };
```

```
 void vecteur3::initialise(float abs=0.,float ord=0.,float haut=0.)
{  // appel de la fonction membre de la classe vecteur
   vecteur::initialise(abs,ord);
   z = haut;
}

 void vecteur3:: homothetie(float val)
{  // appel de la fonction membre de la classe vecteur
   vecteur::homothetie(val);
   z = z*val;
}

 void vecteur3::affiche()
{  // appel de la fonction membre de la classe vecteur
   vecteur::affiche();
   cout<<"z = "<<z<<"\n";
}

 void main()
{
   vecteur3 v, w;
   v.initialise(5,4,3); v.affiche();
   w.initialise(8,2); w.hauteur(7);
   w.affiche();
   cout<<"*****************************\n";
   w.affiche();
   w.homothetie(6); w.affiche();
   getch();
}
```

Exercice XV-3:
A partir de l'exemple précédent, créer un projet. La classe mère sera considérée comme une bibliothèque. Définir un fichier **mere.h** contenant les lignes suivantes :

```
 class vecteur  // classe mère
{
private:
   float x,y;
public: void initialise(float,float);
   void  homothetie(float);
   void affiche();
};
```

Le programme utilisateur contiendra la définition de la classe fille, et le programme principal.

Exercice XV-4:
Dans le programme principal précédent, mettre en œuvre des pointeurs de **vecteur.**

Remarque :
L'amitié n'est pas transmissible: une fonction amie de la classe mère ne sera amie que de la classe fille que si elle a été déclarée amie dans la classe fille.

II- DERIVATION DES CONSTRUCTEURS ET DU DESTRUCTEUR

On suppose la situation suivante :

class A	class B : public A
{ ...	{ ...
public :	public :
A (.....) ; // constructeur	B (.....) ; // constructeur
~A ( ) ; // destructeur	~B( ) ; // destructeur
.....	.....
} ;	} ;

Si on déclare un objet B, seront exécutés
- Le constructeur de A, puis le constructeur de B,
- Le destructeur de B, puis le destructeur de A.

Exemple (à tester) et exercice XV-5:

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

    class vecteur  // classe mère
   {
   private:
      float x,y;
   public:
      vecteur(); // constructeur
      void affiche();
      ~vecteur();     // destructeur
   };

    vecteur::vecteur()
   {
      x=1; y=2; cout<<"Constructeur de la classe mère\n";
   }
    void vecteur::affiche()
   {
      cout<<"x = "<<x<<"  y = "<<y<<"\n";
   }
    vecteur::~vecteur()
   {
      cout<<"Destructeur de la classe mère\n";
   }
```

```
 class vecteur3 : public vecteur // classe fille
{
private :
   float z;
public:
   vecteur3();  // Constructeur
   void affiche();
   ~vecteur3();
};

 vecteur3::vecteur3()
{
   z = 3;
   cout<<"Constructeur de la classe fille\n";
}

 void vecteur3::affiche()
{
   vecteur::affiche();
   cout<<"z = "<<z<<"\n";
}

 vecteur3::~vecteur3()
{
   cout<<"Destructeur de la classe fille\n";
}

 void main()
{
   vecteur3 v; v.affiche();
   getch();
}
```

Lorsque il faut passer des paramètres aux constructeurs, on a la possibilité de spécifier au compilateur vers lequel des 2 constructeurs, les paramètres sont destinés :

Exemple (à tester) et exercice XV-6:
Modifier le programme principal, pour tester les différentes possibilités de passage d'arguments par défaut.

```
#include <iostream.h>
#include <conio.h>
// Héritage simple
    class vecteur  // classe mère
   {
   private:
      float x,y;
   public:
      vecteur(float,float); // constructeur
      void affiche();
      ~vecteur();  // destructeur
   };

    vecteur::vecteur(float abs=1, float ord=2)
   {
      x=abs;y=ord; cout<<"Constructeur  de la classe mère\n";
   }
__
    void vecteur::affiche()
   {
      cout<<"x = "<<x<<"  y = "<<y<<"\n";
   }

    vecteur::~vecteur()
   {
      cout<<"Destructeur de la classe mère\n";
   }

    class vecteur3:public vecteur    // classe fille
   {
   private:
      float z;
   public:
      vecteur3(float, float, float); // Constructeur
      void affiche();
      ~vecteur3();
   };
```

```
 vecteur3::vecteur3(float abs=3, float ord=4, float haut=5)
 :vecteur(abs,ord)
{// les 2 1ers paramètres sont pour le constructeur de la classe mère
   z = haut; cout<<"Constructeur fille\n";
}

 void vecteur3::affiche()
{
   vecteur::affiche(); cout<<"z = "<<z<<"\n";
}

 vecteur3::~vecteur3()
{
   cout<<"Destructeur de la classe fille\n";
}

 void main()
{
   vecteur u; vecteur3 v, w(7,8,9);
   u.affiche();v.affiche(); w.affiche();
   getch();
}
```

Cas du constructeur par recopie (***)

Rappel : Le constructeur par recopie est appelé dans 2 cas :
- Initialisation d'un objet par un objet de même type :
  vecteur a (3,2) ;
  vecteur b = a ;
- Lorsqu'une fonction retourne un objet par valeur :
  vecteur a, b ;
  b = a.symetrique() ;

Dans le cas de l'héritage, on doit définir un constructeur par recopie pour la classe fille, qui appelle le constructeur par recopie de la classe mère.

Exemple (à tester) et exercice XV-7:

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

 // classe mère

    class vecteur
   {
   private:
      float x,y;
   public:
      vecteur(float,float); // constructeur
      vecteur(vecteur &);   // constructeur par recopie
      void affiche();
      ~vecteur();           // destructeur
   };

    vecteur::vecteur(float abs=1, float ord=2)
   {
      x=abs;
      y=ord;
      cout<<"Constructeur de la classe mère\n";
   }

    vecteur::vecteur(vecteur &v)
   {
      x=v.x;
      y=v.y;
      cout<<"Constructeur par recopie de la classe mère\n";
   }

    void vecteur::affiche()
   {
      cout<<"x = "<<x<<"  y = "<<y<<"\n";
   }

    vecteur::~vecteur()
   {
      cout<<"Destructeur de la classe mère\n";
   }
```

```
 class vecteur3:public vecteur // classe fille
{
private:
   float z;
public:
   vecteur3(float, float, float);  // Constructeur
   vecteur3(vecteur3 &);  // Constructeur par recopie
   void affiche();
   ~vecteur3();
};

 vecteur3::vecteur3(float abs=3, float ord=4, float haut=5)
 :vecteur(abs,ord)
{
   z = haut; cout<<"Constructeur de la classe fille\n";
}

 vecteur3::vecteur3(vecteur3 &v)
 :vecteur(v)// appel du constructeur par recopie de la classe vecteur
{
   z = v.z; cout<<"Constructeur par recopie de la classe fille\n";
}

 void vecteur3::affiche()
{
   vecteur::affiche(); cout<<"z = "<<z<<"\n";
}

 vecteur3::~vecteur3()
{
   cout<<"Destructeur de la classe fille\n";
}

 void main()
{
   vecteur3 v(5,6,7); vecteur3 w = v;
   v.affiche(); w.affiche();
   getch();
}
```

III- LES MEMBRES PROTEGES

On peut donner à certaines données d'une classe mère le statut « protégé ».
Dans ce cas, les fonctions membres, et les fonctions amies de la classe fille auront accès aux données de la classe mère :

```
 class vecteur  // classe mère
{
protected:
   float x,y;
public:
   vecteur(float,float); // constructeur
   vecteur(vecteur &);   // constructeur par recopie
   void affiche();
   ~vecteur();           // destructeur
};

 void vecteur3::affiche()
{
   cout<< "x = "<<x<<"  y= "<<y<< "  z = "<<z<<"\n";
}
```

La fonction **affiche** de la classe **vecteur3** a accès aux données **x** et **y** de la classe **vecteur**. Cette possibilité viole le principe d'encapsulation des données, on l'utilise pour simplifier le code généré.

IV- EXERCICES RECAPITULATIFS

On dérive la classe **chaîne** de l'exercice V-10:

```
 class chaine
{
private:
   int longueur; char *adr;
public:
   chaine();
   chaine(char *);
   chaine(chaine &); //constructeurs
   ~chaine();
   void operator=(chaine &);
   int operator==(chaine);
   chaine &operator+(chaine);
   char &operator[](int);
   void affiche();
};
```

La classe dérivée se nomme **chaine_T**.

```
 class chaine_T : public chaine
{
   int Type ;
   float Val ;
public :
   // .......
};
```

Type prendra 2 valeurs : 0 ou 1.
1 si la chaîne désigne un nombre, par exemple « 456 » ou « 456.8 », exploitable par **atof** la valeur retournée sera Val.
0 dans les autres cas, par exemple « BONJOUR » ou « XFLR6 ».

Exercice XV-8: Prévoir pour chaine_T
- un constructeur de prototype **chaine_T() ;** qui initialise les 3 nombres à 0.
- un constructeur de prototype **chaine_T(char *) ;** qui initialise les 3 nombres à 0 ainsi que la chaîne de caractères.
- une fonction membre de prototype **void affiche()** qui appelle la fonction **affiche** de **chaine** et qui affiche les valeurs des 3 nombres.

Exercice XV-9 (***): Prévoir un constructeur par recopie pour chaine_T , qui initialise les 3 nombres à 0.

Exercice XV-10: Déclarer « protected » la donnée **adr** de la classe **chaîne.** Ecrire une fonction membre pour chaine_T de prototype **void calcul()** qui donne les bonnes valeurs à Type, Val.

V- CONVERSIONS DE TYPE ENTRE CLASSES MERE ET FILLE

Règle : La conversion d'un objet de la classe fille en un objet de la classe mère est implicite.
La conversion d'un objet de la classe mère en un objet de la classe fille est interdite.

Autrement dit :

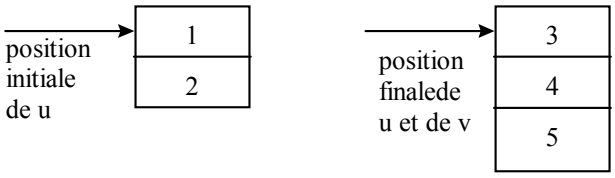
```
vecteur u ;
vecteur3  v;
u = v ;// est autorisée : conversion fictive de v en un vecteur
v = u; // est interdite
```

Exercice XV-11 : Tester ceci avec le programme XV-6

Avec des pointeurs :

```
vecteur *u ;
// réservation de mémoire, le constructeur de vecteur est exécuté
u = new vecteur ;
vecteur3  *v ;
// réservation de mémoire, le constructeur de vecteur3 est exécuté
v = new vecteur3 ;
u = v ; // autorisé, u vient pointer sur la même adresse que v
v = u;  // interdit
delete u ;
delete v ;
```

On obtient la configuration mémoire suivante :



Exemple (à tester) et exercice XV-12 :
Reprendre l'exemple XV-11 avec le programme principal suivant :

```
 void main()
{
   vecteur3 *u = new vecteur3;
   u->affiche();
   vecteur *v = new vecteur;
   v->affiche();
   v = u;  v->affiche();
   delete v ; delete u ; getch();
}
```

Conclusion : quelle est la fonction **affiche** exécutée lors du 2ème appel à **v->affiche()** ?
Le compilateur C++ a-t-il « compris » que v ne pointait plus sur un **vecteur** mais sur un **vecteur3** ?
Grâce à la notion de fonction virtuelle on pourra, pendant l'exécution du programme, tenir compte du type de l'objet pointé, indépendamment de la déclaration initiale.

VI- SURCHARGE DE L'OPERATEUR D'AFFECTATION (***)

Rappels:
- Le C++ définit l'opérateur = pour exprimer l'affectation.
- Il est impératif de le surcharger via une fonction membre, lorsque la classe contient des données de type pointeur pour allouer dynamiquement la mémoire.

A est la classe mère, B est la classe fille, on exécute les instructions suivantes :

B x, y ;
y = x ;

Dans ce cas:
a) Si ni A, ni B n'ont surchargé l'opérateur =, le mécanisme d'affectation par défaut est mis en œuvre.
b) Si = est surchargé dans A mais pas dans B, la surcharge est mise en œuvre pour les données A de B, le mécanisme d'affectation par défaut est mis en œuvre pour les données propres à B.
c) Si = est surchargé dans B, cette surcharge doit prendre en charge la TOTALITE des données (celles de A et celles de B).

Exemple (à tester) et exercice XV-13 :

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

    class vecteur
   {
   private:
      float x,y;
   public:
      vecteur(float,float);
      void affiche();
      void operator=(vecteur &); // surcharge de l'opérateur =
   };

    vecteur::vecteur(float abs=1, float ord=2)
   {
      x=abs; y=ord; cout<<"Constructeur de la classe mère\n";
   }

    void vecteur::affiche()
   {
      cout<<"x = "<<x<<"  y = "<<y<<"\n";
   }
```

```
 void vecteur::operator=(vecteur &v)
{
   cout<<"opérateur affectation de la classe mère\n";
   x = v.x; y = v.y;
}

 class vecteur3 : public vecteur // classe fille
{
private:
   float z;
public:
   vecteur3(float, float, float); // Constructeur
   void operator=(vecteur3 &);    // surcharge de l'opérateur =
   void affiche();
};

 void vecteur3::operator=(vecteur3 &v)
{
   cout<<"operateur affectation de la classe fille\n";
   vecteur *u, *w;
   u = this; w = &v; *u = *w; z = v.z;
}

 vecteur3::vecteur3(float abs=3, float ord=4, float haut=5)
 :vecteur(abs,ord)
{// les 2 1ers paramètres sont pour le constructeur de la classe mère
   z = haut;
   cout<<"Constructeur fille\n";
}

 void vecteur3::affiche()
{
   vecteur::affiche();
   cout<<"z = "<<z<<"\n";
}

 void main()
{
   vecteur3 v1(6,7,8),v2;
   v2 = v1;
   v2.affiche(); getch();
}
```

VII- LES FONCTIONS VIRTUELLES

Les fonctions membres **de la classe mère** peuvent être déclarées ***virtual***.
Dans ce cas, on résout le problème invoqué dans le §IV.

Exemple (à tester) et exercice XV-14 :

Reprendre l'exercice XV-12 en déclarant la fonction **affiche**, **virtual** :

```
 class vecteur  // classe mère
{
private:
   float x,y;
public:
   vecteur(float,float); // constructeur
   virtual void affiche();
   ~vecteur();           // destructeur
};
```

Quelle est la fonction **affiche** exécutée lors du 2ème appel à **v->affiche()** ?
Le programme a-t-il « compris » que v ne pointait plus sur un **vecteur** mais sur un **vecteur3** ?

Ici, le choix de la fonction à exécuter ne s'est pas fait lors de la compilation, mais ***dynamiquement*** lors de l'exécution du programme.

Exemple (à tester) et exercice XV-15 :

Modifier les 2 classes de l'exercice précédent comme ci-dessous :

```
 class vecteur  // classe mère
{
private:
   float x,y;
public:
   vecteur(float,float); // constructeur
   virtual void affiche();
    void message() {cout<<"Message du vecteur\n";}
   ~vecteur();  // destructeur
};

 void vecteur::affiche()
{
   message();cout<<"x = "<<x<<"  y = "<<y<<"\n";
}

 class vecteur3 : public vecteur // classe fille
{
private:
   float z;
public:
   vecteur3(float, float, float);  // Constructeur
   void affiche();
    void message(){cout<<"Message du vecteur3\n";}
   ~vecteur3();
};

 void vecteur3::affiche()
{
   message();
   vecteur::affiche();
   cout<<"z = "<<z<<"\n";
}
```

Remarques :
- Un constructeur ne peut pas être virtuel,
- Un destructeur peut-être virtuel,
- La déclaration d'une fonction membre virtuelle dans la classe mère, sera comprise par toutes les classes descendantes (sur toutes les générations).

Exercice XV-16 :

Expérimenter le principe des fonctions virtuelles avec le destructeur de la classe **vecteur** et conclure.

VIII- CORRIGE DES EXERCICES

Exercice XV-3: Le projet se nomme exvii_3, et contient les fichiers exvii_3.cpp et mere.cpp ou bien mere.obj.

Fichier exvii_3.cpp:

```
#include <iostream.h>
#include <conio.h>
#include "c:\bc5\cours_cpp\teach_cp\chap7\mere.h"

// Construction d'un projet à partir d'une classe mère disponible

    class vecteur3 : public vecteur // classe fille
   {
   private:
      float z;
   public:
      void initialise(float,float,float);
      void  homothetie(float);
       void hauteur(float ha){z = ha;}
      void affiche();
   };

    void vecteur3::initialise(float abs=0.,float ord=0.,float haut=0.)
   {  // fonction membre de la classe vecteur
      vecteur::initialise(abs,ord); z = haut;
   }

    void vecteur3:: homothetie(float val)
   {  // fonction membre de la classe vecteur
      vecteur:: homothetie(val); z = z*val;
   }

    void vecteur3::affiche()
   {  // fonction membre de la classe vecteur
      vecteur::affiche(); cout<<"z = "<<z<<"\n";
   }

    void main()
   {
      vecteur3 v, w;
      v.initialise(5,4,3); v.affiche();
      w.initialise(8,2);   w.hauteur(7);
      w.affiche();
      cout<<"*******************************\n";
      w.affiche();
      w.homothetie(6); w.affiche();
   }
```

Fichier mere.cpp:

```
#include <iostream.h>
#include <conio.h>
#include "c:\bc5\cours_cpp\teach_cp\chap7\mere.h"

    void vecteur::initialise(float abs =0.,float ord = 0.)
   {
      x=abs;
      y=ord;
   }

    void vecteur::homothetie(float val)
   {
      x = x*val;
      y = y*val;
   }

    void vecteur::affiche()
   {
      cout<<"x = "<<x<<"  y = "<<y<<"\n";
   }
```

Exercice XV-4 :

Programme principal :

```
    void main()
   {
      vecteur3 v, *w;
      w = new vecteur3;
      v.initialise(5, 4, 3);v.affiche();
      w->initialise(8,2);w->hauteur(7);
      w->affiche();
      cout<<"*******************************\n";
      w->affiche();
      w->homothetie(6);w->affiche();
      delete w;
   }
```

Exercice XV-8 : Seuls la classe **chaine_T** et le programme principal sont listés

```
    class chaine_T : public chaine
   {
      int Type;
      float Val ;
   public:
      chaine_T(); // constructeurs
      chaine_T(char *);
      void affiche();
   };

 // dans les 2 cas le constructeur correspondant de chaine est appelé
    chaine_T::chaine_T() : chaine()
   {
      Type=0; Val=0;
   }
_
    chaine_T::chaine_T(char *texte) : chaine(texte)
   {
      Type=0; Val=0;
   }

    void chaine_T::affiche()
   {
      chaine::affiche();
      cout<<"Type= "<<Type<<"  Val= "<<Val<<"\n";
   }
```

```
 void main()
{
   chaine a("Bonjour ");
   chaine_T b("Coucou "),c;
   cout<<"a: "; a.affiche();
   cout<<"b: "; b.affiche();
   cout<<"c: "; c.affiche();
   getch();
}
```

Exercice XV-9 : Seules les modifications a été listées

```
 class chaine_T : public chaine
{
private:
   int Type ;
   float Val ;
public:
   chaine_T(); // constructeurs
   chaine_T(char *);
   chaine_T(chaine_T &ch); // constructeur par recopie
   void affiche();
};
```

puis :

```
//constructeur par recopie
  chaine_T::chaine_T(chaine_T &ch) : chaine(ch)
{  // il appelle le constructeur par recopie de chaine
   Type=0; Val=0;
}

 void main()
{
   chaine_T b("Coucou ");
   chaine_T c = b;
   cout<<"b: "; b.affiche();
   cout<<"c: "; c.affiche();
   getch();
}
```

Exercice XV-10 : Seules les modifications ont été listées

```
 void chaine_T::calcul()
{
   Val = atof(adr);    // possible car donnée "protected"
   if(Val!=0)Type = 1;
}
```

puis :

```
 void main()
{
   chaine_T b("Coucou "),c("123"), d("45.9"), e("XFLR6");
   b.calcul(); c.calcul(); d.calcul(); e.calcul();
   b.affiche();c.affiche();d.affiche();e.affiche();
   getch();
}
```

Exercice XV-11 : Seules les modifications ont été listées

```
 void main()
{
   vecteur u;
   vecteur3 v(7,8,9);
   u.affiche();
   v.affiche();
   u=v;
   u.affiche();
   getch();
}
```

COURS et TP DE LANGAGE C++

Chapitre 16

TP de prise en main de C++ Builder

Joëlle MAILLEFERT

joelle.maillefert@iut-cachan.u-psud.fr

IUT de CACHAN

Département GEII 2

INITIATION A L'OUTIL C++ BUILDER

Introduction

Cet outil logiciel BORLAND, basé sur le concept de programmation orientée objet, permet à un développeur, même non expérimenté, de créer assez facilement une interface homme/machine d'aspect « WINDOWS ».
Le programme n'est pas exécuté de façon séquentielle comme dans un environnement classique. Il s'agit de programmation « événementielle », des séquences de programme sont exécutées, suite à des actions de l'utilisateur (clique, touche enfoncée etc…), détectées par WINDOWS.

Ce TP, conçu dans un esprit d'autoformation, permet de s'initier à l'outil, en maîtrisant, en particulier, quelques aspects généraux des problèmes informatiques, à savoir :
- La saisie de données,
- Le traitement,
- La restitution.

Les thèmes proposés fonctionnent avec toutes les versions de « C++ Builder ».

Bibliographie

- L'aide du logiciel,
- BORLAND C++ BUILDER, Gérad LEBLANC, EYROLLES.

1- Création du projet

C++ Builder fonctionne par gestion de projet. Un projet contient l'ensemble des ressources à la construction du fichier exécutable final.
Par défaut, les fichiers utilisateurs sont rangés dans la répertoire
c:\Program Files\Borland\Cbuilder3\Projects (ou Cbuilder5)

Créer un sous-répertoire Travail dans le répertoire Projects de Builder.
Lancer C++ Builder.
Apparaissent
- La fenêtre graphique (***Form1***), dans laquelle seront posés les objets graphiques (menus, boutons etc …),
- Un fichier source ***Unit.cpp***, contenant l'application utilisateur,
- Un fichier source ***Project1.cpp*** qui sera utilisé par WINDOWS pour lancer l'application utilisateur (pour y accéder Voir → Gestionnaire du projet → Double Clique sur ***Project1***),
- un fichier d'en-tête ***Unit1.h*** contenant la déclaration des composants utilisés sous forme de variables globales, ainsi que les fonctions créées. (pour le lire **Fichier → Ouvrir → Unit.h**)
- Dans le bandeau supérieur, un choix de composants graphiques,
- Sur la gauche, «l'inspecteur d'objets », permettant de modifier les propriétés de ces composants graphiques, et de leur associer des événements détectés par WINDOWS.

Modifier le nom du projet : Fichier → Enregistrer le projet sous
- ***Unit.cpp*** devient ***Projects\Travail\Mon_Appli.cpp***
- ***Project1.bpr*** devient ***Projects\Travail\Mon_Proj.bpr***

alors,
- ***Unit.h*** devient automatiquement ***Projects\ Travail\Mon_Appli.h***
- ***Project1.cpp*** devient automatiquement ***Projects\ Travail\Mon_Proj.cpp***

Cliquer dans la fenêtre ***Form1***. Repérer dans l'inspecteur d'objets les 2 propriétés « Name » (nom de la fenêtre dans le programme), et « Caption » (Titre de la fenêtre).
D'une façon générale, la plupart des composants Builder possèdent 2 identifiants :
- « Caption » : nom apparaissant sur l'écran à l'exécution,
- « Name » : nom du composant dans le programme, et donc variable à manipuler.

Modifier le titre de la fenêtre : « Travaux Pratiques »

Exécuter le programme. Le fermer avec la croix en haut à droite de la fenêtre.

Sortir de C++ Builder. Vérifier la présence du fichier exécutable Mon_Proj.exe, dans le répertoire Travail. Le lancer directement et vérifier son fonctionnement.

Revenir à C++ Builder et recharger le projet Mon_Proj.

2- Création d'un menu

On souhaite créer un menu **Fichier**
Saisie
Affichage
Quitter

Dans la boîte à outils « Standard », sélectionner le composant « Main Menu » et le placer dans Form1.
Cliquer sur ce menu, et remplir les différentes cases « Caption ».

Double Cliquer sur la rubrique « Quitter ». On accède alors à la fonction qui sera exécutée lors d'un clique sur cette rubrique. Compléter cette fonction comme ci-dessous.

```
 void __fastcall TForm1::Quitter1Click(TObject *Sender)
{
   Form1-> Close();
}
```

Exécuter le programme et vérifier.
Quitter C++ Builder et vérifier la présence dans le répertoire Travail du fichier ***Mon_Appli.h.*** Le lister et conclure.
Revenir à C++ Builder.

Cliquer dans Form1 et modifier sa couleur, via l'inspecteur d'objets, (choisir le blanc, clWhite). Vérifier en exécutant.

On souhaite maintenant interdire, pour l'instant, l'accès à la rubrique « Affichage » du menu.

Cliquer sur cette rubrique, modifier la propriété « Enabled » en « false » et vérifier en exécutant le programme.

Ajouter maintenant au menu une rubrique « Couleur » :

Fichier	**Couleur**
Saisie	Blanc
Affichage	Gris
Quitter	

Puis en double cliquant sur la rubrique « Blanc », compléter la fonction comme ci-dessous :

```
void __fastcall TForm1::Blanc1Click(TObject *Sender)
{
   Form1->Color = clWhite;
}
```

Enfin en double cliquant sur la rubrique « Gris », compléter la fonction comme ci-dessous :

```
void __fastcall TForm1::Gris1Click(TObject *Sender)
{
   Form1->Color = clGray;
}
```

Vérifier en exécutant.

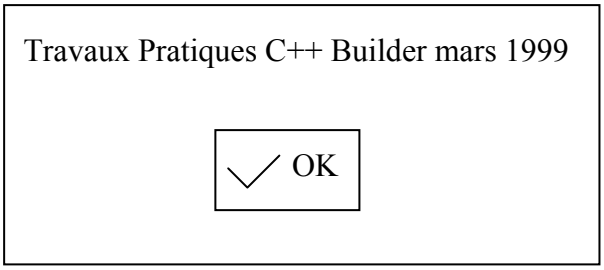
3- Création d'une boîte de dialogue

Ajouter maintenant au menu une rubrique « Outils » :

Fichier	**Couleur**	**Outils**
Saisie	Blanc	A propos de …
Affichage	Gris	
Quitter		

Poser sur la fenêtre ***Form1*** une « Label » (dans les composants « Standard »), avec les propriétés suivantes :
Alignment : taCenter
AutoSize : false
Caption : TP Builder mars 1999
Color : clAqua
Font : Times New Roman / Normal / Taille 12
Height : 80
Left : 150
Name : Label1
Top : 80

Transparent : false
Visible : true
Width : 145
Garder les valeurs par défaut des autres propriétés.
Ajouter au milieu de la Label1 un « BitBtn» (dans les composants « Supplement »), avec les propriétés suivantes :
Name : BitBtn1
Kind : bkOK
Visible : true



Exécuter pour vérifier.

Donner maintenant à Label1 et à Bitbtn1 la propriété Visible = false.

Exécuter pour vérifier.

Cliquer sur la ligne A propos de … du menu et compléter la fonction comme ci-dessous :

```
 void __fastcall TForm1::Aproposde1Click(TObject *Sender)
{
   Label1->Visible = true;
   BitBtn1->Visible = true;
}
```

Exécuter pour vérifier.

Cliquer sur le bouton OK et compléter la fonction comme ci-dessous :

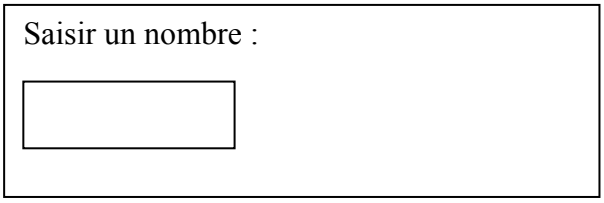
```
 void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
   Label1->Visible = false;
   BitBtn1->Visible = false;
}
```

Exécuter pour vérifier.

4- Création d'une boîte de saisie

Ajouter sur Form1 une « Label » (dans les composants « Standard »), avec les propriétés suivantes :
Alignment : taLeftJustify
AutoSize : false
Caption : Saisir un nombre :
Color : clYellow
Font : Times New Roman / Normal / Taille 10
Height : 80
Left : 50
Name : Label2
Top : 150
Transparent : false
Visible : true
Width : 160

Ajouter au milieu de la Label2 un « Edit» (ab dans les composants « Standard »), avec les propriétés suivantes:
AutoSize : false
Name : Edit1
Color : clWindow
Text :
Visible : true



Exécuter pour vérifier.

Donner maintenant à Label2 et à Edit1 la propriété Visible = false.

Exécuter pour vérifier.

Cliquer sur la rubrique « Saisie » du menu et compléter la fonction comme ci-dessous :

```
 void __fastcall TForm1::Saisie1Click(TObject *Sender)
{
   Label2->Visible = true;
   Edit1->Visible = true;
}
```

Exécuter pour vérifier.

Dans la boîte Edit1, double cliquer sur l'événement **OnKeyDown** et compléter la fonction comme ci-dessous :

```
 void __fastcall TForm1::Edit1KeyDown(TObject *Sender,
                                      WORD &Key, TShiftState Shift)
{
   float r;
   if(Key == VK_RETURN)
   {r = (Edit1->Text).ToDouble();}
}
```

Puis, ajouter au début du source Mon_Appli.cpp la déclaration
#include <vcl\dstring.h>

C++ Builder propose une bibliothèque de manipulations de chaînes de caractères. La classe *AnsiString* y a été créée, munie des méthodes appropriées.
Edit1->Text est de type *AnsiString*.
ToDouble est une méthode qui transforme une chaîne de type *AnsiString* en un nombre réel.

Attention, aucune précaution n'a été prise, ici, en cas d'erreur de saisie (ceci sera étudié à la fin du TP).
Si on effectue une erreur de saisie (par exemple en tapant une lettre), il y aura traitement d'une exception par WINDOWS. Concrètement, pour nous, ceci se traduit, pour l'instant, par un « plantage ».

L'événement **OnKeyDown** détecte l'appui sur une touche du clavier. On valide ici la saisie lorsque l'utilisateur appuie sur la touche « ENTREE ». Celle-ci n'est pas prise en compte dans la saisie.
Exécuter pour vérifier (avec et sans erreur de saisie).

5- Traitement et affichage d'un résultat

Modifier la fonction **Edit1KeyDown** comme ci-dessous :

```
 void __fastcall TForm1::Edit1KeyDown(TObject *Sender,
                                      WORD &Key, TShiftState Shift)
{
   if(Key == VK_RETURN)
   {
      float r = (Edit1->Text).ToDouble();
      Affichage1->Enabled = true; Saisie1->Enabled = false;
   }
}
```

Exécuter pour vérifier.

Déclarer une variable globale comme ci-dessous :
float resultat ;

Puis modifier la fonction **Edit1KeyDown** comme ci-dessous :

```
 void __fastcall TForm1::Edit1KeyDown(TObject *Sender,
                                      WORD &Key, TShiftState Shift)
{
   if(Key == VK_RETURN)
   {
      float r = (Edit1->Text).ToDouble();
      Affichage1->Enabled = true;
      Saisie1->Enabled = false; resultat = r*2 - 54;
   }
}
```

Construire le projet pour vérifier qu'il n'y a pas d'erreur (Menu Projet → Construire).

Aller chercher dans la palette standard le composant « ListBox » (boîte liste)
Le poser sur la fenêtre principale, vers la droite, avec les propriétés suivantes :
Visible = false
Color = clYellow
Height = 65
Left = 280
Top = 24
Width = 120
Name : ListBox1

Modifier la fonction **Saisie1Click** comme ci-dessous

```
 void __fastcall TForm1::Saisie1Click(TObject *Sender)
{
   Label2->Visible = true;
   Edit1->Visible = true;
   ListBox1->Visible = false;
}
```

Cliquer sur la rubrique « Affichage » du menu. Ecrire la fonction de gestion de l'événement OnClick comme ci-dessous :

```
 void __fastcall TForm1::Affichage1Click(TObject *Sender)
{
   AnsiString chaine;
   ListBox1->Visible = true; Edit1->Visible = false;
   Label2->Visible = false;
   chaine= FloatToStrF(resultat,AnsiString::sffGeneral,7,10);
   ListBox1->Clear(); ListBox1->Items->Add(chaine);
   Affichage1->Enabled = false; Saisie1->Enabled = true;
}
```

La fonction **FloatToStrF** permet de convertir un nombre réel en une chaîne de caractères. Voir l'aide pour plus d'informations.
Exécuter pour vérifier.

6- Affichage d'une série de valeurs

Déclarer une variable globale **float tab[3]**, supprimer la variable **resultat**.
Modifier les fonctions **Affichage1Click** et **Edit1KeyDown** comme ci-dessous :

```
 void __fastcall TForm1::Affichage1Click(TObject *Sender)
{
   AnsiString chaine;
   ListBox1->Visible = true; Edit1->Visible = false;
   Label2->Visible = false;
   ListBox1->Clear(); ListBox1->Items->Add("Voici le résultat:");
   for(int i=0;i<3;i++)
   {
      chaine= FloatToStrF(tab[i],AnsiString::sffGeneral,7,10);
      ListBox1->Items->Add(chaine);
   }
   Affichage1->Enabled = false; Saisie1->Enabled = true;
}

 void __fastcall TForm1::Edit1KeyDown(TObject *Sender,
                                      WORD &Key, TShiftState Shift)
{
   if(Key == VK_RETURN)
   {
      float r = (Edit1->Text).ToDouble();
      tab[0] = r*2 - 54; tab[1] = r*3 + 100; tab[2] = r*2.5 - 8.5;
      Saisie1->Enabled = false; Affichage1->Enabled = true;
   }
}
```

Exécuter pour vérifier.

7- Fichiers

Ajouter au menu dans la rubrique « Fichier », une sous-rubrique « Ouvrir » et une sous-rubrique « Enregistrer »
La sous-rubrique « Ouvrir », possède la propriété Enabled = true.
La sous-rubrique « Enregistrer », possède la propriété Enabled = false.

Modifier les fonctions **Affichage1Click** et **Edit1KeyDown** comme ci-dessous :

```
 void __fastcall TForm1::Edit1KeyDown(TObject *Sender,
                                      WORD &Key, TShiftState Shift)
{
   if(Key == VK_RETURN)
   {
      float r = (Edit1->Text).ToDouble();
      Affichage1->Enabled = true; Enregistrer->Enabled = true;
      Saisie1->Enabled = false; Ouvrir1->Enabled = false;
      tab[0] = r*2 - 54; tab[1] = r*3 + 100;
      tab[2] = r*2.5 - 8.5;
   }
}

 void __fastcall TForm1::Affichage1Click(TObject *Sender)
{
   int i;
   AnsiString chaine;
   ListBox1->Visible = true; Edit1->Visible = false;
   Label2->Visible = false;
   ListBox1->Clear(); ListBox1->Items->Add("Voici le résultat:");
   for(i=0;i<3;i++)
   {
      chaine= FloatToStrF(tab[i],AnsiString::sffGeneral,7,10);
      ListBox1->Items->Add(chaine);
   }
   Affichage1->Enabled = false; Saisie1->Enabled = true;
   Enregistrer1->Enabled = true;
}
```

Exécuter pour vérifier.

Aller chercher dans la palette Dialogues le composant « SaveDialog » (boîte de sauvegarde)
Le poser sur la fenêtre principale, avec les propriétés suivantes :

FileName : tp1.dat (nom apparaissant par défaut)
InitialDir : c:\Program Files\Borland\Cbuilder3\Projects\Sei
Filter : *.dat
Name : SaveDialog1.

Cliquer sur la rubrique « Enregistrer » du menu et rédiger la fonction **Enregistrer1Click** comme ci-dessous.

```
 void __fastcall TForm1::Enregistrer1Click(TObject *Sender)
{
   bool res = SaveDialog1->Execute();
}
```

Exécuter pour vérifier.
C'est au programme de sauvegarder réellement le fichier. La variable « res » vaut ***false*** si l'utilisateur a annulé l'opération, ***true*** s'il a réellement sélectionné un fichier.

Ajouter le fichier ***stdio.h*** dans la lister des ***#include***
Modifier fonction **Enregistrer1Click** comme ci-dessous :

```
 void __fastcall TForm1::EnregistrerClick(TObject *Sender)
{
   FILE *f;
   bool res = SaveDialog1->Execute();
   if(res==true)
   {
      f = fopen((SaveDialog1->FileName).c_str(), "wb");
      fwrite(tab,4,3,f); fclose(f) ;
   }
}
```

On utilise ici les fonctions de traitement de fichier classiques de BORLAND C++. La fonction membre ***c_str()*** de la classe ***AnsiString*** permet de convertir une « AnsiString » en tableau de caractères classique, nécessaire ici à la fonction ***fopen***.

Exécuter pour vérifier, en choisissant le nom de fichier proposé par défaut, puis un autre.
Vérifier la présence de ces fichiers dans l'explorateur de WINDOWS.

Aller chercher dans la palette Dialogues le composant « OpenDialog »
Le poser sur la fenêtre principale, avec les propriétés suivantes :

FileName : tp1.dat (nom apparaissant par défaut)
InitialDir : c:\Program Files\Borland\Cbuilder3\Projects\Travail
Filter : *.dat
Name : OpenDialog1.

Cliquer sur la rubrique « Ouvrir » du menu et rédiger la fonction **Ouvrir1Click** comme ci-dessous.

```
 void __fastcall TForm1::Ouvrir1Click(TObject *Sender)
{
   FILE *f;
   bool res = OpenDialog1->Execute();
   if(res==true)
   {
      f = fopen((OpenDialog1->FileName).c_str(), "rb");
      fread(tab,4,3,f); fclose(f); Affichage1->Enabled = true;
   }
}
```

C'est au programme d'ouvrir réellement le fichier. La variable « res » vaut ***false*** si l'utilisateur a annulé l'opération, ***true*** s'il a réellement sélectionné un fichier.

Exécuter pour vérifier.

8- Amélioration de la zone de saisie

Un composant a le « focus d'entrée », lorsqu'il est actif dès la prochaine action sur le clavier.

Modifier la fonction comme **Saisie1Click** ci-dessous :

```
 void __fastcall TForm1::Saisie1Click(TObject *Sender)
{
   Label2->Visible = true; Edit1->Visible = true;
   ListBox1->Visible = false;
   Edit1->SetFocus(); // donne le focus d'entrée à la zone d'édition
}
```

Exécuter pour vérifier.

Modifier la fonction **Edit1KeyDown** comme ci-dessous :

```
 void __fastcall TForm1::Edit1KeyDown(TObject *Sender,
                                      WORD &Key, TShiftState Shift)
{
   if(Key == VK_RETURN)
   {
      try
      {
         float r = (Edit1->Text).ToDouble();
         Affichage1->Enabled = true; Enregistrer->Enabled = true;
         Saisie1->Enabled = false; Ouvrir1->Enabled = false;
         tab[0] = r*2 - 54 tab[1] = r*3 + 100; tab[2] = r*2.5 - 8.5;
      }
          catch(...)
         {ShowMessage("Ce n'est pas un réel");}
   }
}
```

On traite ici l'exception générée par WINDOWS lors d'une erreur de saisie. Si tout se passe bien, le bloc « try » est exécuté, sinon, le bloc « catch » est exécuté.
On découvre ici, au passage la fonction **ShowMessage** qui permet d'afficher très facilement un message soumis à validation sur l'écran.

Exécuter sous Builder pour vérifier, en effectuant une erreur de saisie. WINDOWS génère une erreur. Cliquer sur OK et continuer l'exécution.
Lancer maintenant directement l'exécutable Mon_Proj.exe depuis l'explorateur de WINDOWS, en effectuant une erreur de saisie.
Ca marche !

9- Utilisation d'une ressource système

On souhaite, ici, mettre en œuvre la ressource TIMER.

Sélectionner le composant TIMER dans la palette « système ».
Lui attribuer les propriétés suivantes :

Enabled : false
Interval : 2000 (ce sont des millisecondes)
Name : Timer1

Double Cliquer sur l'événement **OnTimer** et compléter la fonction comme ci-dessous (analogue à Affichage1Click)

```
 void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
   Timer1->Enabled = false;  // inhibition du timer
   AnsiString chaine;
   ListBox1->Visible = true;
   Edit1->Visible = false; Label2->Visible = false;
   ListBox1->Clear(); ListBox1->Items->Add("Voici le résultat:");
   for(int i=0;i<3;i++)
   {
      chaine= FloatToStrF(tab[i],AnsiString::sffGeneral,7,10);
      ListBox1->Items->Add(chaine);
   }
   Affichage1->Enabled = false; Saisie1->Enabled = true;
   Enregistrer->Enabled = true;
}
```

Compléter la fonction **Edit1KeyDown** comme ci-dessous :
On lance dons le timer chaque fois que l'on saisit une valeur dans la boîte de saisie. On obtient dons un affichage 2 secondes après la saisie.

```
 void __fastcall TForm1::Edit1KeyDown(TObject *Sender,
                                      WORD &Key, TShiftState Shift)
{
   if(Key == VK_RETURN)
   {
      try
      {
         float r = (Edit1->Text).ToDouble();
         Affichage1->Enabled = true; Enregistrer->Enabled = true;
         Saisie1->Enabled = false; Ouvrir1->Enabled = false;
         tab[0] = r*2 - 54;
         tab[1] = r*3 + 100;
         tab[2] = r*2.5 - 8.5;
         Timer1->Enabled = true ;
      }
          catch(...)
         {
            ShowMessage("Ce n'est pas un réel");
         }
   }
}
```

Exécuter pour vérifier.

On pourra utiliser le timer, par exemple pour lancer une acquisition sur une carte d'interface, pour provoquer des événements périodiques s'il est toujours validé etc..
Il y a toutefois une limite : la milliseconde !

Remarque : On se prive ici de la rubrique « Affichage » du menu. Il ne faut donc pas faire n'importe quoi !

10- Conclusion

Ce n'est plus qu'une question de temps passé et de bonne documentation …

Si on connaît un peu de programmation orienté objet, quoique la notion de pointeur de structure suffise ici, on peut, à ce stade, analyser le fichier d'en-tête ***Mon_Appli.h,*** et comprendre un peu mieux la construction des composants manipulés dans ce TP.