

CCM Tout sur le C++ : codes source des exemples

Code 2.2 : convertisseur francs-euros	4
Code 2.3 : lecture et écriture <i>via</i> les flux standard	4
Code 2.4 : utilisation de <i>getline()</i>	5
Code 2.5 : lecture caractère par caractère avec <i>get()</i>	6
Code 2.6 : utilisation de <i>put()</i> et de <i>write()</i>	6
Code 2.7 : utilisation des manipulateurs sur un flux de sortie	6
Code 3.1 : prototypes et définitions de fonctions.....	7
Code 3.2 : transmission des arguments par référence	8
Code 3.3 : transmission des arguments par valeur et par pointeur.....	9
Code 3.4 : utilisation de <i>new</i> et de <i>delete</i>	10
Code 3.5 : utilisation des arguments par défaut	11
Code 3.6 : fonction <i>inline</i>	12
Code 3.7 : surcharge d'une fonction	12
Code 4.1 : déclaration d'une classe et définition de ses méthodes	14
Code 4.2 : constructeur et destructeur	15
Code 4.3 : classe <i>Date</i> avec plusieurs constructeurs	16
Code 4.4 : fichier <i>Date.h</i>	17
Code 4.5 : fichier <i>Date.cpp</i>	17
Code 4.6 : prototype et définition de la fonction <i>AugmenteAnnee()</i>	19
Code 4.7 : utilisation d'un constructeur de copie (fichier en-tête <i>Date.h</i>)....	19
Code 4.8 : utilisation d'un constructeur de copie (fichier source <i>Date.cpp</i>)	20
Code 4.9 : surcharge de l'opérateur + (fichier en-tête <i>Date.h</i>).....	21
Code 4.10 : surcharge de l'opérateur + (fichier source <i>Date.cpp</i>)	22
Code 5.1 : fichier en-tête <i>05-01.h</i>	23
Code 5.2 : fichier source <i>05-01.cpp</i>	24
Code 5.3 : déclaration des classes <i>Date</i> et <i>Heure</i> (fichier <i>05-03.h</i>)	27
Code 5.4 : utilisation d'un modèle de fonction (fichier <i>05-04.cpp</i>).....	28
Code 5.5 : utilisation d'un opérateur ami.....	29
Code 5.6 : utilisation d'un opérateur (fonction) modèle et ami et transmission d'un modèle en argument (fichier <i>05-06.h</i>)	30
Code 5.7 : utilisation d'un opérateur (fonction) modèle et ami et transmission d'un modèle en argument (fichier <i>05-06.cpp</i>)	32
Code 5.8 : classe amie	34
Code 6.1 : déclaration d'une classe de base et d'une classe dérivée	35
Code 6.2 : accès à une donnée membre cachée depuis une classe dérivée ..	36

Code 7.1 : affectation d'un pointeur de type Base à un objet de type Derivee	38
Code 7.2 : attention aux méthodes masquées.....	38
Code 7.3 : implémentation d'une liste chaînée.....	39
Code 7.4 : la classe abstraite Noeud.....	44
Code 7.5 : une liste chaînée plus « polymorphique ».....	44
Code 7.6 : implémentation d'une fonction virtuelle pure dans la classe de base.....	47
Code 7.7 : syntaxe de l'héritage multiple.....	47
Code 7.8 : implémentation de deux niveaux d'héritage multiple.....	48
Code 7.9 : héritage virtuel	49
Code 7.10 : héritage virtuel avec initialisation de membres de la classe de base.....	50
Code 8.2 : accès aux éléments d'un <i>vector</i> via l'opérateur surchargé [].....	52
Code 8.3 : accès aux éléments d'un <i>vector</i> via les itérateurs.....	54
Code 8.4 : ajout et suppression d'éléments dans un <i>vector</i>	55
Code 8.5 : opérations sur des <i>vectors</i>	57
Code 8.6 : couper-coller entre deux listes	59
Code 8.7 : ajout et suppression du premier élément d'une liste	60
Code 8.8 : tri, fusion et inversement des éléments de liste	61
Code 8.9 : suppression d'éléments dans une liste.....	63
Code 8.10 : utilisation d'une pile.....	65
Code 8.11 : utilisation d'une file.....	65
Code 8.12 : utilisation d'une file d'attente à priorité.....	66
Code 8.13 : utilisation d'un map.....	67
Code 8.14 : utilisation de la classe <i>multimap</i>	70
Code 9.1 : algorithme de génération et de stockage d'éléments dans un conteneur	71
Code 9.2 : algorithmes de copie	72
Code 9.3 : algorithme d'échange d'éléments	73
Code 9.4 : algorithme de recherche et de remplacement	74
Code 9.5 : algorithme de rotation.....	75
Code 9.6 : algorithme de permutation.....	75
Code 9.7 : algorithme d'inversion	76
Code 9.8 : algorithme de réorganisation aléatoire.....	76
Code 9.9 : algorithmes d'itération	77
Code 9.10 : algorithme de décompte d'éléments.....	77
Code 9.11 : algorithme de recherche d'éléments.....	78
Code 9.12 : algorithmes de recherche de séquence spécifique	78
Code 9.13 : algorithme de recherche de doublons	79

Code 9.14 : algorithme de tri.....	79
Code 9.15 : algorithme de tri partiel.....	80
Code 9.16 : algorithme de positionnement du n-ième élément.....	80
Code 9.17 : algorithmes de détermination du maximum et du minimum....	81
Code 10.1 : Hello World	81

Code 2.2 : convertisseur francs-euros

```
1 : #include<iostream>
2 : using namespace std; //std = espace de noms par défaut
3 :
4 : int main()
5 : {
6 :     const double indice = 6.55957; //valeur en francs d'un euro
7 :     float prix, euros, francs;
8 :     char monnaie = 0;
9 :
10:    cout << "Saisissez la valeur à convertir suivie de sa monnaie:\n";
11:
12:    cin >> prix;    //on lit la valeur saisie au clavier
13:    cin >> monnaie; //on lit l'initiale de la monnaie
14:
15:    switch (monnaie){ //Selon la monnaie
16:    case 'f':        //on convertit en euros
17:        francs = prix;
18:        euros = prix / indice;
19:        break;
20:    case 'e':        //ou on convertit en francs
21:        francs = prix * indice;
22:        euros = prix;
23:        break;
24:    default:
25:        euros = francs = 0;
26:        break;
27:    }
28:    //on affiche le résultat
29:    cout << francs << " francs = ";
30:    cout << euros << " euros\n";
31: }
```

Code 2.3 : lecture et écriture *via* les flux standard

```
1 : #include<iostream>
2 : #include<string>
3 : using namespace std;
4 : int main()
```

```

5 : {
6 :   char nom[50]; //le nom est stocké dans un tableau de caractères
7 :   string prenom; //le prénom est stocké dans une
8 :                       //variable de type string
9 :   string metier;
10:
11:   cout << "Saisissez votre nom:\t";
12:   cin >> nom;
13:   cout << "\nSaisissez votre prénom:\t";
14:   cin >> prenom;
15:
16:   string resultat = "\n\nBonjour " + prenom + " " + nom;
17:   resultat += '\n'; // équivalent à resultat=resultat+'\n'
18:
19:   cout << resultat;
20:
21: }

```

Code 2.4 : utilisation de *getline()*

```

1 : #include<iostream>
2 : #include<string>
3 : using namespace std;
4 :
5 : int main()
6 : {
7 :   string nom, prenom; //le nom complet est stocké dans
8 :                       //2 variables de type string
9 :   cout << "Saisissez votre nom:\t";
10:   getline (cin, nom);
11:   cout << "\nSaisissez votre prénom:\t";
12:   getline (cin, prenom);
13:
14:   cout << "Bonjour " << prenom << " " << nom <<endl;
15:
16:   string message = "Bienvenue";
17:
18:   unsigned int i;
19:   for (i=0; i<message.length(); i++) {
20:     cout << "Lettre " << i << " = " << message[i] << endl; }

```

```
21: }
```

Code 2.5 : lecture caractère par caractère avec get()

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

int main()
{
    char car1, car2;

    cout << "Saisissez votre nom: ";
    cin.get(car1).get(car2); // cin.get est appelée 2 fois
    cout << "Votre nom commence par: " << car1 ;
    cout << car2 << endl; //endl purge le flux
    cin.ignore(80, '\n');
    //on ignore les caractères suivants (80 maxi) sur la ligne
}
```

Code 2.6 : utilisation de put() et de write()

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

int main() {

    char* chaine = "Bonjour les amis!";
    cout.write(chaine,7); // affiche "Bonjour"
    cout.put('\n'); //on écrit un retour à la ligne
    cout.write(chaine, strlen(chaine)) ; // affiche toute la chaîne
    return 0;
}
```

Code 2.7 : utilisation des manipulateurs sur un flux de sortie

```
#include <iostream>
#include <iomanip>

using namespace std;
```

6/82

```

int main()
{
    cout << "On affiche le booléen true sous forme textuelle: \t\t";
    cout << boolalpha << true << endl;
    cout << "On repasse en mode binaire pour ";
    cout << "afficher le booléen false: \t";
    cout << noboolalpha << false << endl;
    cout << "Hex permet aussi d'afficher les booléens ";
    cout << "en mode binaire: \t";
    cout << hex << false << endl;
    cout << "On écrit le nombre 63 en hexadécimal: \t\t\t\t";
    cout << hex << 63 << endl;
    cout << "En majuscules: \t\t\t\t\t\t\t\t";
    cout << uppercase << 63 << endl;
    cout << "On repasse en base 10 pour afficher 63: \t\t\t";
    cout << dec << 63 << endl;
    cout << "Avec le signe: \t\t\t\t\t\t\t\t";
    cout << showpos << 63 << endl;
    cout << "On affiche un flottant sur 15 caractères minimum, \n";
    cout << "en remplissant les vides avec des *: \t\t\t";
    cout << setfill('*') << setw(15) << 3.151592 << endl;
    cout << "On recommence avec un alignement à gauche,\n";
    cout << "sans le signe: \t\t\t\t\t\t\t\t";
    cout << left << noshowpos << setw(15) << 3.151592 << endl;
}

```

Code 3.1 : prototypes et définitions de fonctions

```

#include<iostream>
using namespace std;
#include"definition.h" // inclusion de prototypes et
//définitions extérieures
void Carre(int); //prototype de Carre()

//déclaration et définition de la fonction principale
int main()
{
    int nombre;
    cout << "Saisissez un nombre:" << endl;

```

```

    cin >> nombre;
    Carre(nombre);           //appel de la fonction Carre()
    Affiche(nombre);        //appel de la fonction Affiche()
}

/*****définition de carre()*****/
void Carre(int i)
{
    int j; //déclaration d'un objet local
    j=i*i;
    cout << "Le carré de "<< i << " est " << j << endl;
}
/*****/

```

Code 3.2 : transmission des arguments par référence

```

#include<iostream>
using namespace std;

//prototype de Incremente() :
void Incremente(int valeur, int &reference);

int main()
{
    int v, r;

    cout << "Saisissez la 1ère valeur: ";
    cin >> v;
    cout << "\nSaisissez la 2ième valeur: ";
    cin >> r;
    Incremente (v,r);
    cout << "\n\nAprès exécution de Incremente(), il reste en mémoire: ";
    cout << v << " pour la 1ère valeur\n";
    cout << "Par contre, la 2ième qui avait été transmise par référence";
    cout << " a la valeur: " << r << endl;

}

// Définition de Incremente() :
void Incremente(int valeur, int &reference)

```

```

{
    valeur++; //on incrémente une copie de l'argument valeur
    reference++; //on incrémente l'argument réel
    cout << "\nDans la fonction Incrémente() les valeurs deviennent: ";
    cout << valeur << " et " << reference ;
}

```

Code 3.3 : transmission des arguments par valeur et par pointeur

```

#include<iostream>
using namespace std;

//prototype de Incrémente() :
void Incrémente(int valeur1, const int* pointeur);

int main()
{
    int var1, var2;

    cout << "Saisissez la 1ère valeur: ";
    cin >> var1;
    cout << "\nSaisissez la 2ième valeur: ";
    cin >> var2;
    int* pointeur = &var2 ;
    Incrémente (var1, pointeur);
    cout << "Après exécution de Incrémente(), var1=" << var1 ;
    cout << "\nvar2 = " << var2 << endl;

}

// Définition de Incrémente() :
void Incrémente(int v, const int* p)
{
    v++; //on incrémente une copie de l'argument
    int var=*p; //on crée une copie locale de l'argument
    // constant
    var++; //on incrémente la copie locale
}

```

9/82

```
    cout << "Incremente() produit les deux valeurs " ;  
    cout << v << " et " << var << endl;  
}
```

Code 3.4 : utilisation de new et de delete

```
#include<iostream>  
using namespace std;  
  
//prototype de Incremente() :  
void Incremente(int *valeur1, const int *valeur2);  
  
int main()  
{  
  
    int *var1=new int;    //on réserve la mémoire dans le tas  
    int *var2=new int;  
  
    if((var1==0)|(var2==0)) {  
        cout << "Erreur, il n'y a plus de mémoire disponible dans le tas!";  
        return;    // si le tas est saturé, on termine le programme  
    }  
  
    cout << "Saisissez la 1ère valeur: ";  
    cin >> *var1;    //on stocke la valeur de var1 dans  
                    // l'emplacement réservé  
    cout << "\nSaisissez la 2ième valeur: ";  
    cin >> *var2;    //idem pour var2  
  
    Incremente (var1, var2);  
    cout << "Après exécution de Incremente(), var1= " ;  
    cout << *var1 << " et var2 = " << *var2 << endl;  
  
    delete var1;    //on libère la mémoire occupée  
    delete var2;    //dans le tas  
}  
  
// Définition de Incremente() :  
void Incremente(int *v1, const int *v2)    //transmission de  
                                           // 2 pointeurs dont un constant
```

```

{
    *v1=*v1+1;    //on incrémente la valeur stockée dans le tas
    int var=*v2; //on crée une copie locale de la valeur
                //stockée dans le tas
    var++;       //on incrémente la copie locale
    cout << "\nIncrémenté produit les deux valeurs ";
    cout << *v1 << " et " << var << endl;
}

```

Code 3.5 : utilisation des arguments par défaut

```

#include<iostream>
using namespace std;

void Affiche(int resultat, int base=10); //10 est la base par
                                        // défaut

int main()
{
    int nbre;           //déclarations de variables
    char reponse;
    int base;

    cout << "Saisissez l'entier à afficher: ";
    cin >> nbre;
    Affiche(nbre);      // Affichage en base 10 par défaut
    cout << "\nVoulez-vous l'afficher dans une autre base? (o/n)";
    cin >> reponse;    // lecture de la réponse
    switch(reponse) {
        case 'o':      //si l'utilisateur a répondu o
            cout << "\nSaisissez la base:";
            cin >> base; //lecture de la base
            Affiche(nbre, base); //affichage dans le format demandé
            break;
        default:      //sinon on termine
            return 0;
    }
}

```

```

}

void Affiche(int r, int b)
{
    switch(b) {
        case 10:
            cout << "\nVoici le résultat en base 10: " << r;
            break;
        case 16:
            cout << "\nVoici le résultat en base 16: " << hex << r ;
            break;
        case 8:
            cout << "\nVoici le résultat en base 8: " << oct << r;
            break;
        default:
            cout << "\nerreur" << endl;
    }
}

```

Code 3.6 : fonction inline

```

#include<iostream>
using namespace std;

inline int fac(int);

int main()
{
    cout << "Le factoriel de 6 est: " << fac(6) << endl;
}

inline int fac(int n)
{
    return(n<2)?1:n*fac(n-1);
//si n<2 on renvoie 1 sinon on renvoie n*fac(n-1)
}

```

Code 3.7 : surcharge d'une fonction

```

#include<iostream>

```

```

using namespace std;

//Prototypes de la fonction surchargée
void Affiche(int);
void Affiche(const char*);
void Affiche(double);
void Affiche(long);
void Affiche(char);

int main()
{
    int i=2;    //déclaration et initialisation d'un entier
    Affiche(i); //Pas de problème, on utilise le 1er prototype

    Affiche("Il fait beau aujourd'hui"); //Pas de problème, on
                                         //utilise le 2ième prototype

    float f=6.359;
    Affiche(f); //le compilateur convertit le float en
               // double, on utilise le 3ième prototype

    short s=9;
    Affiche(s); //le compilateur convertit le short en int,
               //on utilise le 1er prototype

    Affiche('c'); // Pas de problème, on utilise
                 //le 5ième prototype
    Affiche("il neige"); //Pas de problème, on utilise
                        // le 2ième prototype
}

void Affiche(int x)
{
    cout << "Affichage d'un entier: " << x << endl;
}
void Affiche(const char* x)
{
    cout << "Affichage d'une chaîne: " << x << endl;
}
void Affiche(double x)

```

```

{
    cout << "Affichage d'un double: " << x << endl;
}
void Affiche(long x)
{
    cout << "Affichage d'un long: " << x << endl;
}
void Affiche(char x)
{
    cout << "Affichage d'un caractère: " << x << endl;
}

```

Code 4.1 : déclaration d'une classe et définition de ses méthodes

```

#include<iostream> //Pour les entrées/sorties
#include<string> //Pour les chaînes
using namespace std;

class Date //Définition du nouveau type Date
{
    int jour,annee; //Les données membres privées par défaut
    string mois;

public: //partie publique
    void DefinirDate(int,string,int); //méthode d'accès
    void LireDate(); //méthode d'accès
}; //ne pas oublier le point virgule

/****Définition de la méthode d'accès DefinirDate()****/

void Date::DefinirDate(int j,string m,int a)
{
    //on définit la date à partir des arguments transmis
    jour=j;
    mois=m;
    annee=a;
}

```

```

/****Définition de la méthode d'accès LireDate()****/

void Date::LireDate()
{
    //on affiche la valeur des données jour mois annee
    cout << "Date: " << jour << " " << mois << " " << annee ;
}

int main()                //Fonction principale
{
    Date Date_du_jour;    //création d'une instance de classe
    Date_du_jour.DefinirDate(3,"novembre",2009); //initialisation des
données membres
    Date_du_jour.LireDate();
//On affiche la valeur des données membres
}

```

Code 4.2 : constructeur et destructeur

```

#include<iostream>        //Pour les entrées/sorties
#include<string>          //Pour les chaînes
using namespace std;

class Date                //Définition du nouveau type Date
{
    int jour,annee;        //Les données membres
    string mois;          //privées par défaut

public:                   //partie publique
    Date(int,string,int);    //Constructeur
    ~Date();                 //Destructeur
    void DefinirDate(int,string,int); //méthode d'accès
    void LireDate() const;   //méthode d'accès

};

/****Définition du constructeur****/
Date::Date(int jourInitial,string moisInitial,int anneeInitial)
{
    jour=jourInitial;    //Initialisation des données membres
}

```

```

    mois=moisInitial;
    annee=anneeInitial;
}

/****Définition du destructeur****/
Date::~Date()          //pas d'argument ni de valeur retournée
{ }

void Date::DefinirDate(int j,string m,int a)
{ jour=j; mois=m; annee=a; }

void Date::LireDate() const
{
    cout << "Date: " << jour << " " << mois << " " << annee ;
}

int main()              //Fonction principale
{
    int j,a;
    string m;
    cout << "Saisissez la date (ex:5 novembre 2009): ";
    cin >> j;
    cin >> m;
    cin >> a;
    Date Date_du_jour(j,m,a); //création d'une instance de classe
    Date_du_jour.LireDate(); //On affiche la valeur des données membres
}

```

Code 4.3 : classe Date avec plusieurs constructeurs

```

class Date {
    int jour,annee;          //Les données membres
    string mois;           //privées par défaut

public:
    Date(int,string,int);  //jour, mois, année
    Date(int,string);     //jour, mois, année courante
    Date(int);            //jour, mois et année courants
    Date();               //date courante par défaut
    ~Date();              //Destructeur
}

```

```

void DefinirDate(int,string,int); //méthode d'accès
void LireDate() const ;          //méthode d'accès

};

```

Code 4.4 : fichier Date.h

```

#include<iostream>      //Pour les entrées/sorties
using namespace std;

class Date
{
private:
    int jour, mois, annee;
    static Date date_par_defaut; //membre statique
public:
    Date(int j=0,int m=0,int a=0);
    ~Date();
    void DefinirDate(int j,int m,int a)
    { jour=j; mois=m; annee=a; }
    void LireDate() const
    { cout<<"Date: "<<jour<<" "<<mois<<" "<<annee<<endl; }
//fonctions membres statiques
    static void DefinirDefaut(int,int,int);
    static void LireDefaut();
};

```

Code 4.5 : fichier Date.cpp

```

#include"Date.h"      //Inclusion de la déclaration de Date

/**Initialisation du membre statique***/
Date Date::date_par_defaut(1,1,2009);

****Définition du constructeur****/
Date::Date(int j,int m,int a)
{
//Initialisation des données membres :
jour=j ? j:date_par_defaut.jour;
mois=m ? m:date_par_defaut.mois;
}

```

```

    annee=a ? a:date_par_defaut.annee;
}

/****Définition du destructeur****/
Date::~Date() { }

/*Définition de la fonction membre statique DéfinirDefaut()*/
void Date::DefinirDefaut(int jj,int mm,int aa)
{ Date::date_par_defaut=Date(jj,mm,aa); }

/*Définition de la fonction membre statique LireDefaut()*/
void Date::LireDefaut( )
{ cout << Date::date_par_defaut.jour << "/" <<
Date::date_par_defaut.mois;
  cout << "/" << Date::date_par_defaut.annee ; }

int main()      //Fonction principale
{
    int j, m, a;
    char reponse;

    cout << "La date par défaut est: ";
    Date::LireDefaut();
    cout << "\nVoulez vous en changer? (o/n): ";
    cin >> reponse;

    if(reponse=='o') {
        cout << "\nSaisissez la nouvelle date par défaut (ex:5 12 2010): ";
        cin >> j;
        cin >> m;
        cin >> a;
        Date::DefinirDefaut(j,m,a);
        cout << "\nLa nouvelle date par défaut est: " ;
        Date::LireDefaut(); //On vérifie la valeur du membre statique
    }
    Date date_standard=Date(); //on crée une instance
                                //avec les valeurs par défaut
    Date date_perso=Date(5,3,2009); //on crée une instance
                                //avec les arguments transmis
    cout << "\nVersion standard: ";

```

```
date_standard.LireDate();
cout << "\nVersion perso: ";
date_perso.LireDate();
}
```

Code 4.6 : prototype et définition de la fonction AugmenteAnnee()

```
Date& AugmenteAnnee()(int n); //prototype à inclure dans la
                             //déclaration de classe

/*****Définition de AugmenteAnnee()*****/

Date& Date::AugmenteAnnee(int n)
{
    if(jour==29 && mois==2 && !leapyear(annee+n){
//s'il s'agit du 29 février et que annee+n n'est
// pas bissextile
        jour=1; //on modifie aussi le jour et le mois
        mois=3;
    }
    annee+=n; //forme abrégée de annee=annee+n
    return *this; //on retourne une référence de l'objet
}
```

Code 4.7 : utilisation d'un constructeur de copie (fichier en-tête Date.h)

```
#include<iostream> //Pour les entrées/sorties
using namespace std;

class Date
{
private:
    int *jour, *mois, *annee;
public:
    Date(int,int,int); //Constructeur par défaut
    Date (const Date &); //constructeur de copie
    ~Date(); //Destructeur
```

```

    /* Méthodes d'accès */
    void DéfinirDate(int j,int m,int a)
    { *jour=j; *mois=m; *annee=a; }
    void LireDate() const
    { cout<<*jour<<" "<<*mois<<" "<<*annee<<endl; }
    int LireJour() const { return *jour; }
    int LireMois() const { return *mois; }
    int LireAnnee() const { return *annee; }
};

```

Code 4.8 : utilisation d'un constructeur de copie (fichier source Date.cpp)

```

#include"Date.h" //Inclusion de la déclaration de Date

/****Définition du constructeur****/
Date::Date(int j,int m,int a)
{
/*réservation de l'espace mémoire dans le tas, les valeurs des membres
sont ensuite stockées aux adresses correspondantes*/
    jour=new int; *jour=j;
    mois=new int; *mois=m;
    annee=new int; *annee=a; //
}

/****Définition du constructeur de copie****/
Date::Date(const Date &source) //l'objet source est protégé par const
{
/*réservation de l'espace mémoire dans le tas puis copie des valeurs aux
adresses correspondantes*/
    jour=new int; *jour=source.LireJour();
    mois=new int; *mois=source.LireMois();
    annee=new int; *annee=source.LireAnnee();
}

/****Définition du destructeur****/
Date::~Date()
{
    delete jour; jour=0; //libération de la mémoire
    delete mois; mois=0;
}

```

```

    delete annee; annee=0;
}

int main()
{
    cout << "\nNous créons l'objet Date_initiale= 1 1 2009 ";
    Date Date_initiale(1,1,2009);
    cout << "\nNous le copions dans l'objet Copie_date ";
    Date Copie_date=Date_initiale ;
    //équivalent à Date Copie_date(Date_initiale);

    cout << "\nNous changeons maintenant Date_initiale en 31 12 2010 " <<
endl;
//modification de l'objet initial :
    Date_initiale.DefinirDate(31,12,2010);
    cout << "Nous obtenons les deux objets : \n" ;
    cout << "Date_initiale = ";
        Date_initiale.LireDate();
    cout << "Alors que Copie_date a toujours la valeur: ";
    Copie_date.LireDate();
    cout << "\nCQFD" << endl;
}

```

Code 4.9 : surcharge de l'opérateur + (fichier en-tête *Date.h*)

```

#include<iostream>      //Pour les entrées/sorties
using namespace std;

class Date
{
private:
    int jour, mois, annee;
public:
    Date(int,int,int);      //Constructeur par défaut
    ~Date();                //Destructeur
    void DefinirDate(int j,int m,int a)
        { jour=j; mois=m; annee=a; }
    void LireDate() const
        { cout<<jour<<" "<<mois<<" "<<annee<<endl; }
    int LireJour() const { return jour; }
}

```

21/82

```

    int LireMois() const { return mois; }
    int LireAnnee() const { return annee; }
    Date operator+ (const Date &);
};

```

Code 4.10 : surcharge de l'opérateur + (fichier source *Date.cpp*)

```

#include "Date.h" //Inclusion de la déclaration de Date

/****Définition du constructeur****/
Date::Date(int j,int m,int a)
{ jour=j; mois=m; annee=a; }

/****Définition du destructeur****/
Date::~Date() { }

/****Définition de l'opérateur +****/
Date Date::operator + (const Date &source)
{
    int j, m, a;
    j=jour + source.LireJour();
    m=mois + source.LireMois();
    a=annee + source.LireAnnee();
    if(j>30) {
        j-=30;
        m+=1;
    }
    if(m>12) {
        m-=12;
        a+=1;
    }
    return Date(j,m,a);
}

int main()
{
    cout << "Nous créons l'objet Date_initiale= 1 1 2009 ";
    Date Date_initiale(1,1,2009);
    cout << "\nNous lui ajoutons 30 jours et 6 mois via l'objet Duree 30 6
0" << endl;
}

```

22/82

```

    Date Duree(30,6,0);
    //on ajoute la durée à la date initiale :
    Date Date_finale=Date_initiale + Duree ;
    cout << "Nous obtenons la nouvelle date: ";
    Date_finale.LireDate();
}

```

Code 5.1 : fichier en-tête 05-01.h

```

#include<iostream>      //Pour les entrées/sorties
using namespace std;

class Date //déclaration de la classe Date
{
private:
    int jour, mois, annee;
public:
    Date();           //constructeur par défaut
    Date(int,int,int); //Constructeur
    ~Date();          //Destructeur
    /** Méthodes d'accès ***/
    void DefinirDate(int j,int m,int a)
        { jour=j; mois=m; annee=a; }
    void LireDate() const
        { cout<<jour<<" "<<mois<<" "<<annee<<endl; }
    int LireJour() const { return jour; }
    int LireMois() const { return mois; }
    int LireAnnee() const { return annee; }

    //opérateur = surchargé pour Date
    Date operator=(const Date);
};

//déclaration du modèle avec son type paramètre (T) :
template <class T>
class Tableau //déclaration de la classe "paramètre"
{
private:
    T * p_tab; //déclaration du pointeur de tableau p_tab
    //autre élément important d'un tableau, sa taille :

```

```

    int taille;

public:
    //Constructeur
    Tableau(int taille);
    Tableau(const Tableau &source); //Constructeur de copie
    ~Tableau() { delete [] p_tab; } //Destructeur

    //surchage de l'opérateur = :
    Tableau& operator=(const Tableau&);

    //surcharges de l'opérateur [] :
    T& operator[] (int index) { return p_tab[index]; }
    const T& operator[](int index) const
        { return p_tab[index]; }
    //méthode d'accès
    int LireTaille() const { return taille; }
};

```

Code 5.2 : fichier source 05-01.cpp

```

1 : #include"05-01.h" //déclarations Date et modèle
2 :
3 : int taille_defaut=4;
4 :
5 : /**Constructeurs et destructeur de Date***/
6 : Date::Date()
7 :     { jour=0; mois=0; annee=0; }
8 : Date::Date(int j,int m,int a)
9 :     { jour=j; mois=m; annee=a; }
10:
11: Date::~Date() { }
12:
13: /****Définition de l'opérateur = surchargé pour Date****/
14:
15: Date Date::operator=(const Date source)
16: {
17:     if (this == &source) //si les deux objets sont égaux
18:         return *this; //on renvoie tout de suite l'objet
19:     jour = source.LireJour(); //sinon on copie jour

```

```

20:     mois = source.LireMois(); //puis mois
21:     annee = source.LireAnnee(); //et enfin annee
22:     return source;
23: }
24: /*Définition du constructeur du modèle de
25:  classe Tableau*/
26: template <class T>
27: //initialisation de la donnée membre taille :
28: Tableau<T>::Tableau(int t=taille_defaut):taille(t)
29: {
30:     p_tab = new T[t];
31: }
32:
33: /****Définition du constructeur de copie du modèle****/
34:
35: template <class T>
36: Tableau<T>::Tableau(const Tableau &source)
37: /*le constructeur de copie reçoit en argument l'adresse
38:  d'un tableau source protégé par const*/
39: {
40: /*on détermine la taille en fonction de celle du tableau
41:  source :*/
42:     taille = source.LireTaille();
43:     p_tab = new T[taille]; //on réserve la mémoire
44:                             // appropriée dans le tas
45:     for(int i=0; i<taille; i++) //on recopie les éléments
46:         p_tab[i]=source[i];
47: }
48:
49: /****Définition de l'opérateur = pour le modèle****/
50: template <class T>
51: Tableau<T>& Tableau<T>::operator=(const Tableau &source)
52: {
53:     if (this == &source) //si les 2 tableaux sont égaux
54:         return *this; //on renvoie tout de suite l'objet cible
55:     // et la suite de l'instruction if n'est pas exécutée
56:     delete [] p_tab; //on supprime l'objet cible
57: /*on détermine sa taille en fonction de celle du tableau
58:  source :*/
59:     taille = source.LireTaille(); //

```

```

60: /*on réserve de la mémoire dans le tas pour "recréer"
61:  l'objet cible : */
62:  p_tab = new T[taille];
63: /*on recopie chaque élément de l'objet source dans
64:  l'objet cible : */
65:  for (int i=0; i<taille; i++)
66:      p_tab[i] = source[i];
67:  return *this;    //on renvoie l'objet cible
68: }
69:
70: int main()
71: {
72:  Tableau<Date> Tab_date; //on crée un tableau de dates
73:  Date *p_date;    //on déclare un pointeur d'objet Date
74:  /** Les tableaux peuvent être volumineux, c'est pourquoi
75:  il est préférable de les créer dans le tas et de les
76:  manipuler via des pointeurs **/
77:
78: cout<<"Saisissez les dates importantes à enregistrer (ex 5 12 2009):
"<<endl;
79: //pour chaque élément du tableau :
80:  for (int i=0; i<Tab_date.LireTaille(); i++)
81:  {
82:      int j, m, a;
83:      cout << "\t\tDate "<<"i" <<": ";
84:      cin >> j;          //on récupère la valeur
85:      cin >> m;          //saisie par l'utilisateur
86:      cin >> a;
87:      p_date = new Date(j, m, a); //on réserve la mémoire
88:                               //pour le nouvel élément
89:      Tab_date[i] = *p_date; //on stocke sa valeur à
90:                          // l'emplacement réservé
91:  }
92:
93: cout << "Voici le tableau des dates enregistrées: \n" ;
94:  for (int k=0; k<Tab_date.LireTaille(); k++)
95:  {
96:      Tab_date[k].LireDate() ;
97:  }
98: }

```

Code 5.3 : déclaration des classes *Date* et *Heure* (fichier *05-03.h*)

```
#include<iostream>      //Pour les entrées/sorties
using namespace std;

class Heure; //Déclaration de la classe Heure

class Date //déclaration et définition de la classe Date
{
private:
    int jour, mois, annee;
public:
    Date();           //constructeur par défaut
    Date(int,int,int); //Constructeur
    ~Date();          //Destructeur

    //méthodes d'accès
    void DéfinirDate(int j,int m,int a)
        { jour=j; mois=m; annee=a; }
    int LireJour() const { return jour; }
    int LireMois() const { return mois; }
    int LireAnnee() const { return annee; }

    //opérateur = surchargé pour Date
    Date operator=(const Date);

    //opérateur ami (voir section suivante):
    friend ostream& operator<< (const Date&, const Heure&);
};

class Heure //définition de la nouvelle classe Heure
{
private:
    int heure, minute;
public:
    Heure(int,int); //Constructeur
    ~Heure();       //Destructeur

    //méthodes d'accès
    void DéfinirHeure(int h,int m){ heure=h; minute=m; }
    int LireHeure() const { return heure; }
};
```

27/82

```

    int LireMinute() const { return minute; }

    //opérateur ami (voir section suivante):
    friend ostream& operator<< (const Date&, const Heure&);
};

```

Code 5.4 : utilisation d'un modèle de fonction (fichier 05-04.cpp)

```

#include"05-03.h" //Inclusion des déclarations Date et Heure

/****Constructeurs et destructeur de Date****/
Date::Date()
    { jour=0; mois=0; annee=0; }
Date::Date(int j,int m,int a)
    { jour=j; mois=m; annee=a; }

Date::~Date() { }

/****Constructeur et destructeur de Heure****/
Heure::Heure(int h,int m)
    { heure=h; minute=m; }

Heure::~Heure() { }

/****opérateur << surchargé pour afficher une date ****/
ostream& operator<< (ostream& sortie, const Date une_date)
{
    sortie << une_date.LireJour() << " " << une_date.LireMois() << " " <<
une_date.LireAnnee();
    return sortie;
}

/****opérateur << surchargé pour afficher une heure ****/
ostream& operator<< (ostream& sortie, const Heure une_heure)
{
    sortie << une_heure.LireHeure() << "h" << une_heure.LireMinute();
    sortie << "mn";
    return sortie;
}

```

```

//déclaration du modèle avec son type paramètre (T) :
template <class T>
//déclaration de la fonction "paramétrée" :
void Affiche(T x)
{
    cout << x << endl; //on affiche l'objet à condition que
} //ce dernier implémente l'opérateur surchargé <<

int main()
{
    cout << "\nNous créons l'objet Date_initiale = 1 1 2009 ";
    Date Date_initiale(1,1,2009);
    cout << "\nPuis l'objet Heure_initiale = 12h30 " << endl;
    Heure Heure_initiale(12,30);

    cout << "\nNous affichons maintenant ces deux objets avec Affiche()\n"
<< endl;
    cout << "Date:\t";
    Affiche(Date_initiale);
    cout << "Heure:\t";
    Affiche(Heure_initiale);
}

```

Code 5.5 : utilisation d'un opérateur ami

```

1 : #include"05-03.h" //Inclusion de la déclaration de Date
2 : // et Heure
3 :
4 : /*Recopier les définitions suivantes dans le Code 5.4*/
5 : /****Constructeurs et destructeurs de Date et Heure****/
6 : /****opérateur << surchargé pour afficher une date ****/
7 : /****opérateur << surchargé pour afficher une heure ****/
8 :
9 : //déclaration du modèle avec ses paramètres (T1 et T2):
10: template <class T1, class T2>
11: //déclaration de la fonction "paramétrée" :
12: void Affiche(T1 x, T2 y)
13: {
14: //on affiche les 2 objets :
15: cout << "le " << x << " à " << y << endl;

```

```

16: }
17:
18: int main()
19: {
20:     int j,m,a,hh,mm;
21:     cout << "\nSaisissez la date du RV (ex: 2 12 2010): ";
22:     cin >> j;
23:     cin >> m;
24:     cin >> a;
25:     Date date_RV(j,m,a); //on crée une instance de Date
26:     cout << "\nSaisissez l'heure du RV (ex: 14 45): ";
27:     cin >> hh;
28:     cin >> mm;
29:     Heure heure_RV(hh,mm); //puis une instance de Heure
30:     cout << "\nSeule une fonction utilisant un opérateur d'affichage
ami des 2 classes peut afficher la date et l'heure du RV:" << endl;
31:     Affiche(date_RV, heure_RV);
32: }

```

Code 5.6 : utilisation d'un opérateur (fonction) modèle et ami et transmission d'un modèle en argument (fichier 05-06.h)

```

1 :#include<iostream>          //Pour les entrées/sorties
2 :using namespace std;
3 :
4 ://déclaration de la classe Date
5 :class Date
6 :{
7 :private:
8 :     int jour, mois, annee;
9 : public:
10:     Date();                //constructeur par défaut
11:     Date(int,int,int);     //Constructeur
12:     ~Date();               //Destructeur
13:
14:
15:     //méthodes d'accès
16:     void DefinirDate(int j,int m,int a)
17:         { jour=j; mois=m; annee=a; }
18:     int LireJour() const { return jour; }
19:     int LireMois() const { return mois; }

```

30/82

```

20:  int LireAnnee() const { return annee; }
21:
22:  //opérateur = surchargé pour Date
23:  Date operator=(const Date);
24:
25: };
26:
27: //déclaration du modèle avec son type paramètre (T)
28: template <class T>
29:
30: //déclaration de la classe "paramètre"
31: class Tableau
32: {
33: private:
34:   T * p_tab; //déclaration du pointeur de tableau p_tab
35:   int taille; //autre élément important d'un tableau
36:
37: public:
38:   //Constructeur avec argument par défaut
39:   Tableau(int taille);
40:   //Constructeur de copie
41:   Tableau(const Tableau &source);
42:   //Destructeur
43:   ~Tableau() { delete [] p_tab; }
44:
45:   //surcharge de l'opérateur = :
46:   Tableau& operator=(const Tableau&);
47:   //surcharge de l'opérateur [] :
48:   T& operator[] (int index) { return p_tab[index]; }
49:   const T& operator[](int index) const
50:     { return p_tab[index]; }
51:   int LireTaille() const { return taille; }
52:
53:   //fonction amie
54:   friend ostream& operator<< (const Date&, const Tableau<T>&);
55: };

```

Code 5.7 : utilisation d'un opérateur (fonction) modèle et ami et transmission d'un modèle en argument (fichier 05-06.cpp)

```
1 : #include"05-06.h"
2 : const int taille_defaut=4;
3 : /**Constructeurs et destructeur de Date***/
4 :Date::Date()
5 : { jour=0; mois=0; annee=0; }
6 :Date::Date(int j,int m,int a)
7 :{ jour=j; mois=m; annee=a; }
8 :
9 :Date::~Date() { }
10:
12:/**opérateur << surchargé pour afficher une date ***/
13: ostream& operator<< (ostream& sortie, const Date une_date)
14:{
15:  sortie << une_date.LireJour() << " ";
16:  sortie << une_date.LireMois() << " ";
17:  sortie << une_date.LireAnnee();
18:  return sortie;
19:}
20:
21: /**Définition de l'opérateur = surchargé pour Date**/
22:
23: Date Date::operator=(const Date source)
24: {
25:  if (this == &source) //si les deux objets sont égaux
26:    return *this;    //on renvoie tout de suite l'objet
27:  jour = source.LireJour(); //sinon on copie jour
28:  mois = source.LireMois(); //puis mois
29:  annee = source.LireAnnee(); //et enfin annee
30:  return source;
31: }
32:
33:/*Définition du constructeur du modèle
34:  de classe Tableau*/
35: template <class T>
36: Tableau<T>::Tableau(int t=taille_defaut):taille(t)
37: {
38:  p_tab = new T[t];
```

```

39: }
40:
41: /****La fonction operator<< amie et modèle****/
42: template <class T>
43: ostream& operator<< (ostream& sortie, const Tableau<T>& un_tableau)
44: {
45:     for(int i=0; i<un_tableau.LireTaille(); i++)
46:         sortie << i+1 << " : " << un_tableau[i] << endl;
47:     return sortie;
48: }
49:
50: //prototype de l'affichage d'un tableau de dates
51: void ListeRV(Tableau<Date>& un_tableau);
52:
53: int main()
54: {
55:     Tableau<Date> mon_tableau;
56:     ListeRV(mon_tableau);
57: }
58:
59: /*définition de la fonction d'affichage d'un tableau
60: de dates*/
61: void ListeRV(Tableau<Date>& un_tableau)
62: {
63:     Date *p_date;
64:     cout << "Saisissez les dates de Rendez-vous (ex 5 12 2009): " <<
endl;
65:     //on initialise les éléments du tableau
66:     for (int i=0; i<un_tableau.LireTaille(); i++)
67:     {
68:         int j, m, a;
69:         cout << "\t\tDate "<< i+1 <<": ";
70:         cin >> j;
71:         cin >> m;
72:         cin >> a;
73:         p_date = new Date(j, m, a); //on réserve la mémoire
74:         un_tableau[i] = *p_date; //on stocke sa valeur à
75:                                 // l'emplacement réservé
76:         delete p_date; //le tableau contient une copie
77:     }

```

```
78:     //maintenant on les affiche
79:     cout << un_tableau << endl ;
80: }
```

Code 5.8 : classe amie

```
#include<iostream>     //Pour les entrées/sorties
using namespace std;

/* Déclaration de la classe principale */
class UneClasse
{
    friend class Amie;
    // Toutes les méthodes de Amie sont amies

    int i;             // Donnée privée de UneClasse

public:
    UneClasse(void) { i=5; return i; } //constructeur
    int LireObjet() const { return i; }
};

/* Déclaration de la classe amie */
class Amie
{
public:
    void AfficherObjet(UnClasse un_objet)
    // on affiche la donnée privée de objet
    { cout << un_objet.LireObjet(); }
};

int main()
{
    UneClasse objet; //on crée un objet de type UneClasse
    Amie a;         //on crée un objet de type Amie
    cout << "L'objet a la valeur: ";
    a.AfficherObjet(objet); //on affiche la valeur de objet via a
}
```

Code 6.1 : déclaration d'une classe de base et d'une classe dérivée

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

/* Déclaration de la classe de base */
class Base
{
private:
    //membres privés
    int val_privee;
public:
    Base(){ cout << "Le constructeur de Base s'exécute\n";
           val_protected=5;
    }
    ~Base(){ cout << "Le destructeur de Base s'exécute\n"; }
    void Afficher()
        { cout << "La fonction Afficher() de Base s'exécute\n"; }
protected:
    //membres protégés
    int val_protected ;
};

/* Déclaration de la classe dérivée */
class Derivee : public Base
{
private:

public:
    Derivee()
    { cout << "Le constructeur de Derivee s'exécute\n"; }
    ~Derivee()
    { cout << "Le destructeur de Derivee s'exécute\n"; }
    //Méthodes d'accès
    void Lire() const
    { cout << "Lecture de val_protected depuis la classe Derivee: ";
      cout << val_protected << endl; }
};

int main()
{
```

```
    Derivee objet_derivee;    //déclare un objet Derivee
//on accède à une fonction de la classe de base :
    objet_derivee.Afficher();
//on accède à un membre protégé de la classe de base :
    objet_derivee.Lire();
}
```

Code 6.2 : accès à une donnée membre cachée depuis une classe dérivée

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

//Déclarations des classes

class Base
{
public:
    string i;
    Base(){}
    ~Base(){}
    //méthodes d'accès
    void Definir();
    string Lire();
};

class Derivee : public Base
{
public:
    string i;
    //méthodes d'accès
    string Lire();
    void Definir();
    Derivee(){}
    ~Derivee(){ }
};

//définitions des fonctions
```

```

void Base::Definir()
{ i="C'est la donnée publique de la classe de base"; }
void Derivee::Definir()
{ i="C'est la donnée publique de la classe dérivée\n"; }

string Base::Lire()
{
    return i; // renvoie la valeur du membre i
}

string Derivee::Lire()
{
    return i; // renvoie la valeur du membre i
}

int main(void)
{
    cout << "On crée un objet de type Derivee" << endl;
    Derivee objet_derivee;

    cout << "On initialise le membre i de Base\n";
    objet_derivee.Base::Definir();
    cout << "On initialise le membre i de Derivee\n";
    objet_derivee.Derivee::Definir();

    cout << "On affiche la valeur renvoyée par la fonction Lire()";
    cout << " de objet_derivee: \n";
    //on affiche le membre i de Derivee :
    cout << objet_derivee.Lire();
    cout << "Pour accéder au membre i masqué de la classe de base,\n";
    cout << "il faut appeler explicitement la fonction Lire() de Base:
\n";
    // on affiche l'entier i de la classe Base :
    cout << objet_derivee.Base::Lire() << endl;
    cout << "Ou afficher explicitement le membre masqué: \n";
    cout << objet_derivee.Base::i << endl;
    return 0;
}

```

Code 7.1 : affectation d'un pointeur de type Base à un objet de type Derivee

```
1 :#include <iostream.h>
2 :/** Insérer ici les déclarations des classes Base et Derivee du Code
3 : */
4 :
5 : int main()
6 : {
7 :     //on affecte un pointeur de Base à un objet Derivee
8 :     Base *objet_derivee = new Derivee;
9 :     objet_derivee->Afficher(); //on accède à une fonction de la
10:    classe de base
11:    //objet_derivee->Lire(); //erreur, Lire n'est pas membre de Base
12:    delete objet_derivee;
13: }
```

Code 7.2 : attention aux méthodes masquées

```
1 :#include <iostream>
2 :using namespace std;
3 :
4 :/* Déclaration de la classe de base */
5 :class Base
6 :{
7 :private:
8 :    int val_privée;
9 :public:
10:    Base(){}
11:    virtual ~Base(){}
12:    void Afficher() const; //cette fonction est surchargée
13:    void Afficher(int); //dans la classe de base
14:};
15:
16:/* Déclaration de la classe dérivée */
17:class Derivee : public Base
18:{
19:private:
20:
21:public:
```

```

22:  Derivee(){
23:  ~Derivee(){
24:  //Méthodes d'accès
25:  void Afficher() const; //une seule version définie
26:}; // dans la classe dérivée
27:
28:/* Définition des 2 fonctions Afficher de Base **/
29:
30:void Base::Afficher() const
31: { cout<<"La fonction Afficher() de Base s'exécute\n"; }
32:void Base::Afficher(int i)
33: {
34:     val_privee=i;
35:     cout<<"Valeur de val_privee avec Afficher(int) de Base: ";
36:     cout << val_privee << endl; }
37:
38:/* Définition de la fonction Afficher de Derivee **/
39:
40:void Derivee::Afficher() const
41:{
42:     cout << "La fonction Afficher() de Derivee s'exécute\n";
43:}
44:int main()
45:{
46:     //on déclare un objet de type Derivee
47:     Derivee objet_derivee;
48:     //on accède à la version sans argument
49:     objet_derivee.Afficher();
50:     //objet_derivee.Afficher(5); //erreur
51:     objet_derivee.Base::Afficher();
52:     objet_derivee.Base::Afficher(5);
53:}

```

Code 7.3 : implémentation d'une liste chaînée

```

1 : #include <iostream>
2 :
3 : /*****Classe Donnees*****/
4 : class Donnees //classe des objets à enregistrer
5 : { //dans la liste

```

```

6 : public:
7 :   Donnees (int val):valeur_objet(val){} //constructeur
8 :           //avec initialisation du membre valeur_objet
9 :   ~Donnees (){}
10: /*Comparer() permet à deux objets Donnees de se comparer
11: afin de respecter un certain ordre dans la liste*/
12:   int Comparer(const Donnees &);
13: /*Afficher() affiche la valeur de l'objet*/
14:   void Afficher() { std::cout << valeur_objet << std::endl; }
15: private:
16:   int valeur_objet;
17: };
18: enum { plus_petit, plus_grand, egal};
19:
20: int Donnees::Comparer(const Donnees &autre_objet)
21: { /*cette fonction permet de comparer les valeurs de 2 objets
Donnees*/
22:   if (valeur_objet < autre_objet.valeur_objet)
23: /*si la valeur du noeud à insérer est inférieure à celle
24: du noeud appelant :*/
25:     return plus_petit;
26:   if (valeur_objet > autre_objet.valeur_objet)
27: /*si la valeur du noeud à insérer est supérieure
28: à celle du noeud appelant : */
29:     return plus_grand;
30:   else
31:     return egal;
32: }
33:
34: //Déclarations préalables
35: class Noeud;
36: class PremierNoeud;
37: class Queue;
38: class NoeudInterne;
39:
40: /******Classe Noeud*****/
41: class Noeud
42: {
43: private:
44: public:

```

```

45:   Noeud(){} //constructeur par défaut
46:   virtual ~Noeud(){} //destructeur virtuel
47: /*Inserer() reçoit un objet Donnees et le place dans la liste*/
48: //fct virtuelle pure
49:   virtual Noeud * Inserer(Donnees * un_objet) = 0;
50: /*Afficher() affiche la valeur d'un objet Donnees dans la liste*/
51: //fonction virtuelle pure
52: virtual void Afficher() = 0;
53: };
54: /*****Classe Noeud Interne*****/
55: class NoeudInterne: public Noeud
56: {
57: private:
58:   Donnees *mes_donnees;
59:   Noeud *noeud_suivant; //pointeur du noeud
60:                               //suivant dans la liste
61:
62: public:
63:   NoeudInterne(Donnees *un_objet, Noeud *suivant);
64:   virtual ~NoeudInterne(){ delete noeud_suivant; delete mes_donnees;
65: }
66:   virtual Noeud * Inserer(Donnees *un_objet);
67:   virtual void Afficher()
68:   { mes_donnees->Afficher(); noeud_suivant->Afficher(); }
69: };
70: NoeudInterne::NoeudInterne(Donnees * un_objet, Noeud * suivant):
71: mes_donnees(un_objet),noeud_suivant(suivant)
72: {
73: }
74:
75: Noeud * NoeudInterne::Inserer(Donnees * un_objet)
76: {
77:   int resultat = mes_donnees->Comparer(*un_objet);
78: /*le noeud interne compare la valeur du nouveau noeud avec la
79: sienne*/
80:   switch(resultat)
81:   {
82:     case egal: //si plus grand ou égal
83:     case plus_grand: //on insère avant le noeud courant

```

```

83:     {
84:         NoeudInterne * noeud_donnee =
85:             new NoeudInterne(un_objet, this);
86:         return noeud_donnee;
87:     }
88:
89:     case plus_petit: //si plus petit on transmet au noeud suivant
90:         noeud_suivant = noeud_suivant->Inserer(un_objet);
91:         return this;
92:     }
93:     return this;
94: }
95:
96: /*****Classe Queue*****/
97: class Queue : public Noeud
98: {
99: private:
100:
101: public:
102:     Queue(){} //constructeur par défaut
103:     virtual ~Queue(){} //destructeur virtuel
104:     virtual Noeud * Inserer(Donnees * un_objet);
105:     virtual void Afficher() { }
106: };
107:
108: Noeud * Queue::Inserer(Donnees * un_objet)
109: { /*création d'un nouveau NoeudInterne immédiatement
110:     avant le noeud de queue*/
111:     NoeudInterne * noeud_donnee = new NoeudInterne(un_objet, this);
112:     return noeud_donnee; //on renvoie le pointeur du nouveau noeud
113: }
114:
115: /*****Classe ListeChaine*****/
116: class ListeChaine : public Noeud
117: {
118: private:
119:     Noeud * noeud_suivant; //pointeur du noeud
120:                                     //suivant dans la liste
121:
122: public:

```

```

123:   ListeChainee();           //constructeur par défaut
124:   virtual ~ListeChainee() { delete noeud_suisant; }
125:   virtual Noeud * Inserir(Donnees * un_objet);
126:   virtual void Afficher() { noeud_suisant->Afficher(); }
127:
128: };
129: /**Définitions des fonctions de la classe ListeChainee***/
130: ListeChainee::ListeChainee() //constructeur du noeud de tête
131: {
132:   noeud_suisant = new Queue; //le noeud de fin est créé
133: } //en même temps que le noeud de tête:
134: //on obtient une liste chaînée vide
135:
136: Noeud * ListeChainee::Inserir(Donnees * un_objet)
137: /*la fonction inserer() reçoit le pointeur de l'objet à insérer dans
138: la liste...*/
139: {
140:   noeud_suisant = noeud_suisant->Inserir(un_objet);
141: /*... et le transmet à la fonction Inserir associée au noeud
suisant*/
142:   return this;
143: }
144: /*****Fonction principale*****/
145: int main()
146: {
147:   Donnees *p_donnee; //pointeur d'un objet Donnees
148:   int val;           //une valeur
149:   ListeChainee lc; //et on crée la liste chaînée lc
150:
151:   for (;;) //tant que l'utilisateur ne tape pas 0
152:   {
153:     std::cout << "Quelle valeur voulez-vous ajouter \x85 la liste ?";
154:     std::cout << " (0 si fin): ";
155:     std::cin >> val; //on lit la valeur saisie
156:     if (!val) //si val est nulle
157:       break; //on interrompt la boucle
158:     p_donnee = new Donnees (val); /*sinon on crée un nouvel
159: objet Donnees de valeur val dans le tas*/
160:     lc.Inserir(p_donnee); //et on insère cet objet
161:   } //dans la liste

```

```
162:
163:  std::cout << "\n\n";
164:  lc.Afficher(); //on affiche les valeurs de la liste
165:  std::cout << "\n\n";
166:  return 0;
167: }
```

Code 7.4 : la classe abstraite Noeud

```
Class Noeud
{
private:

public:
    Noeud(){}
    virtual ~Noeud(){}
//les 2 fonctions virtuelles pures
    virtual Noeud * Inserir(Donnees *un_objet) = 0;
    virtual void Afficher() = 0;
};
```

Code 7.5 : une liste chaînée plus « polymorphique »

```
/******Classe Donnees******/
class Donnees //classe abstraite
{
public:
    Donnees(int val):valeur_objet(val){}
    virtual ~Donnees(){}
    int Comparer(const Donnees &);
    virtual void Afficher() = 0; //fonction virtuelle pure
protected:
    int valeur_objet;
};

/***Définition de la fonction Comparer de Donnees***/
int Donnees::Comparer(const Donnees & autre_objet)
{
    if (valeur_objet < autre_objet.valeur_objet)
        return plus_petit;
}
```

```

        if (valeur_objet > autre_objet.valeur_objet)
            return plus_grand;
        else
            return egal;
    }

    /*****Classe DonneeEntiere*****/
    class DonneeEntiere : public Donnees
    {
    public:
        //le constructeur initialise val :
        DonneeEntiere(int val) : Donnees(val) {}
        virtual ~DonneeEntiere() {}
        virtual void Afficher();
    private:
    };

    /*Implémentation de la fonction virtuelle pure Afficher*/
    void DonneeEntiere::Afficher()
    {
        cout << "La valeur de l'entier est " << valeur_objet << endl;
    }

    /*****Classe DonneeGraphique*****/
    class DonneeGraphique : public Donnees
    {
    public:
        DonneeGraphique(int val) : Donnees(val) {}
        virtual ~DonneeGraphique() {}
        virtual void Afficher();
    private:
    };

    /*Implémentation de la fonction virtuelle pure Afficher*/
    void DonneeGraphique::Afficher()
    {
        cout << "(" << valeur_objet << " étoiles): ";
        for ( int i = 0; i < valeur_objet; i++)
            cout << "*";
        cout << endl;
    }

```

```

}

/*****Fonction principale*****/
int main()
{
    Donnees * p_donnees; //on crée un objet Donnees de
                        // pointeur p_donnees
    int val; //valeur à stocker dans la liste
    int type_donnees; //type de la valeur val
    ListeChainee lc; //on crée un objet listechaine

    for (;;) //tant que l'utilisateur de saisi pas 0
    {
        cout << "[1] Entier, [2] Etoiles, [0] Quitter: ";
        cin >> type_donnees; //on lit la valeur saisie
        if (!type_donnees) //si 0
            break; //on termine
        switch(type_donnees)
        {
            case 1: //si 1
                cout << "Quelle valeur voulez-vous ajouter à la liste ? ";
                cin >> val; //on lit la valeur entière
                p_donnees = new DonneeEntiere(val);
                break;
            case 2: //si 2
                cout << "Quelle quantité d'étoiles voulez-vous ajouter à la liste ?
";
                cin >> val; //on lit toujours une valeur entière mais
                //elle représente cette fois une quantité
                p_donnees = new DonneeGraphique(val);
        }

        lc.Inserer(p_donnees); //on stocke dans la liste
    }
    cout << "\n\n";
    lc.Afficher(); //on affiche les valeurs stockées dans
    cout << "\n\n"; //la liste par ordre croissant
    return 0;
}

```

Code 7.6 : implémentation d'une fonction virtuelle pure dans la classe de base

```
1 :class Donnees
2 :{
3 :public:
4 :   Donnees(int val):valeur_objet(val){}
5 :   virtual ~Donnees(){} //destructeur virtuel
6 :   int Comparer(const Donnees &);
7 :   virtual void Afficher() = 0; //fonction virtuelle pure
8 :protected:
9 :   int valeur_objet;
10:};
11://implémentation d'un affichage de base :
12:void Donnees::Afficher()
13:{
14:   cout << "\nVoici la valeur de vos données: \n";
15:}
```

Code 7.7 : syntaxe de l'héritage multiple

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

class Enfant //classe dérivée
{
public:
    void Marcher(); //il marche
    void Rire(); //il rit
protected:
private:
};

class Oiseau
{
public:
    void Chanter(); //il chante
    void Voler(); //il vole
};
```

47/82

```
//la classe Cherubin hérite d'Enfant et de Oiseau :  
class Cherubin : public Enfant, public Oiseau  
{  
};
```

Code 7.8 : implémentation de deux niveaux d'héritage multiple

```
#include <iostream>  
using namespace std;  
  
class Creature  
{  
public:  
    Creature():age(3){}  
    void Dormir(){}  
    int LireAge() const { return age; }  
private:  
    int age;  
};  
  
class Enfant : public Creature  
{  
public:  
    void Marcher(){}  
    void Rire(){}  
protected:  
private:  
};  
  
class Oiseau : public Creature  
{  
public:  
    void Chanter(){}  
    void Voler(){}  
};  
  
class Cherubin : public Enfant, public Oiseau  
{
```

```

};

int main()
{
    Cherubin g;
    cout << g.LireAge(); // ambigu !
    return 0;
}

```

Code 7.9 : héritage virtuel

```

#include <iostream>
using namespace std;

class Creature
{
public:
    Creature():age(3)
    { cout << "Exécution du constructeur de Créature\n"; }
    ~Creature()
    { cout << "Exécution du destructeur de Créature\n"; }
    void Dormir(){}
    int LireAge() const { return age; }
private:
    int age;
};

class Enfant : virtual public Creature
{
public:
    Enfant()
    { cout << "Exécution du constructeur de Enfant\n"; }
    ~Enfant()
    { cout << "Exécution du destructeur de Enfant\n"; }
    void Marcher(){}
    void Rire(){}
protected:
private:
};

```

49/82

```

class Oiseau : virtual public Creature
{
public:
    Oiseau()
        { cout << "Exécution du constructeur de Oiseau\n"; }
    ~Oiseau()
        { cout << "Exécution du destructeur de Oiseau\n"; }
    void Chanter(){}
    void Voler(){}
};

class Cherubin : public Enfant, public Oiseau {};

int main()
{
    cout << "On crée un objet Cherubin" << endl;
    Cherubin amour;
    cout << "Notre chérubin a " << amour.LireAge() << " ans\n";
    return 0;
}

```

Code 7.10 : héritage virtuel avec initialisation de membres de la classe de base

```

#include <iostream>
using namespace std;

class Creature //classe de base
{
public:
    Creature(int un_age):age(un_age){}
    void Dormir(){}
    int LireAge() const { return age; }
private:
    int age;
};

//Enfant hérite de la classe de base virtuelle Creature
class Enfant : virtual public Creature
{

```

```

public:
//le constructeur initialise un_age et nombre_dents
    Enfant(int un_age, int combien_de_dents):
        Creature(un_age),
        nombre_dents(combien_de_dents){}
    void Marcher(){}
    void Rire(){}
    int LireDents() const { return nombre_dents; }
private:
    int nombre_dents;
};

//Oiseau hérite de la classe de base virtuelle Creature
class Oiseau : virtual public Creature
{
public:
//le constructeur initialise un_age et poids
    Oiseau(int un_age, int le_poids):
        Creature(un_age), poids(le_poids){}
    void Chanter(){}
    void Voler(){}
    int LirePoids() const { return poids; }
private:
    int poids;
};

//Cherubin hérite de Creature, Enfant et Oiseau :
class Cherubin : public Enfant, public Oiseau
{
public:
    Cherubin(int un_age, int le_poids, int combien_de_dents):
        // on initialise la classe de base Enfant
        Enfant(un_age, combien_de_dents),
        // on initialise la classe de base Oiseau
        Oiseau(un_age, le_poids),
        // on initialise la classe de base virtuelle Creature!
        Creature(un_age){}

};

```

```

int main()
{
    int son_age = 4;
    int son_poids = 20;
    int ses_dents = 10;

    /*on crée un objet Cherubin en initialisant les
    trois variables */
    Cherubin amour(son_age, son_poids, ses_dents);
    cout<<"Notre chérubin a "<<amour.LireAge()<<" ans\n";
    cout << "Il pèse " << amour.LirePoids() << " kg\n";
    cout << "Et il a " << amour.LireDents() << " dents\n";

}

```

Code 8.2 : accès aux éléments d'un *vector* via l'opérateur surchargé []

```

#include <iostream>
#include <vector> // définition du modèle de classe vector
#include <stdexcept> // pour l'exception out_of_range using namespace std;

typedef vector<int> v_entiers;

//prototype de la fonction AfficherVector
template<class T, class A>
void AfficherVector(const vector<T, A>& v);

// Définition de la fonction AfficherVector
template<class T, class A>
void AfficherVector(const vector<T, A>& v)
{
    // On parcourt le vecteur via l'opérateur []
    cout << "\nVoici les éléments:\n";
    for (vector<int>::size_type i=0; i < v.size(); ++i)
        cout << v[i] << ", ";

    //on affiche la valeur du premier élément :
    cout << "\nValeur de front() = " << v.front();
    //on affiche la valeur du dernier élément :

```

```

    cout << "\tValeur de back() = " << v.back();

    cout << "\n\n";
}

int main()
{
    cout << "On crée le vecteur d'entiers v_ent(3)";
    cout << " avec 3 éléments" << "\n";
    v_entiers v_ent(3);

    cout << "On affecte une valeur à chaque élément via l'opérateur []" <<
endl;
    for (vector<int>::size_type i = 0; i < v_ent.size(); ++i)
        v_ent[i] = i*i ;

    cout << "On tente d'affecter une valeur à un élément";
    cout << " via la fonction at()" << endl;
    try
    {
        v_ent.at((v_entiers::size_type)4) = 50;
        // Erreur, hors de l'intervalle admis !
    }
    // accès hors de l'intervalle admis?
    catch(out_of_range)
    {
        //si oui, on affiche le message :
        cout << "Index hors de l'intervalle admis" << endl;
    }
    AfficherVector(v_ent);

    // Augmente de 5 la taille de v_ent et ajoute 100 à la fin
    v_ent.resize(5, 100);
    cout << "On resize(5, 100) notre vecteur v_ent\n";
    cout << "On essaie maintenant d'accéder au 4ième élément toujours";
    cout << " via la fonction at()" << endl;
    try
    {
        v_ent.at((v_entiers::size_type)4) = 50; //OK
    }
}

```

```

        cout << "Index hors de l'intervalle admis" << endl;
    }
    AfficherVector(v_ent);
}

```

Code 8.3 : accès aux éléments d'un vector via les itérateurs

```

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

typedef vector<int> v_entiers;

template<class T, class A>
void AfficherVector(const vector<T, A>& v);

int main()
{
    cout << "On définit un vecteur d'entiers de 3 éléments\n";
    v_entiers v_ent(3);

    cout << "On affecte une valeur à chaque élément de v_ent";
    cout << " via un itérateur" << endl;
    int i = 0;
    v_entiers::iterator itor ;
    for (itor = v_ent.begin(); itor != v_ent.end(); ++itor)
        *itor = i * i++;
    cout << "Voici sa valeur: " << *++itor << endl;
    AfficherVector(v_ent);
}

//Définition de la fonction AfficherVector
template<class T, class A>
void AfficherVector(const vector<T, A>& v)
{
    // On parcourt le vecteur via un itérateur
    cout << "\nEléments affichés via un itérateur:\n";
    for (typename vector<T, A>::const_iterator itor = v.begin();
        itor != v.end(); ++itor)
        cout << *itor << ", ";
}

```

54/82

```

cout << "\n";
// On parcourt le vecteur via un itérateur inverse
cout << "\nÉléments affichés via un itérateur inverse:\n";
for (typename vector<T, A>::const_reverse_iterator r_itor = v.rbegin();
     r_itor < v.rend(); ++r_itor)
    cout << *r_itor << ", ";
cout << "\n\n";
}

```

Code 8.4 : ajout et suppression d'éléments dans un vector

```

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

typedef vector<int> v_entiers;
typedef vector<int>::iterator iter_vEnt;

template<class T, class A>
void AfficherVector(const vector<T, A>& v);

int main()
{
    cout << "On définit un vecteur d'entiers avec 5 éléments\n";
    v_entiers v_ent(5);

    cout << "On affecte une valeur à chaque élément de v_ent";
    cout << " via l'opérateur index []:\n";
    for (vector<int>::size_type i = 0; i < v_ent.size(); ++i)
        v_ent[i] = 5 * i;

    AfficherVector(v_ent);

    cout << "On insère un élément avec insert(v_ent.begin());";
    cout << " + 1, 50)\nNotre conteneur devient:\n";
    iter_vEnt itor = v_ent.insert(v_ent.begin() + 1, 50);
    AfficherVector(v_ent);
    cout << "Voici l'élément courant : " << *itor << "\n\n";

    cout << "On insère 5 éléments avec insert(v_ent.end(), 5, 30):\n";
}

```

55/82

```

v_ent.insert(v_ent.end(), 5, 30);
AfficherVector(v_ent);

cout << "On supprime un élément dans VInt, on obtient:\n";
v_ent.erase(v_ent.begin() + 3);
AfficherVector(v_ent);

cout << "On supprime 3 éléments dans VInt:\n";
v_ent.erase(v_ent.begin() + 3, v_ent.begin() + 6);
AfficherVector(v_ent);

cout << "On insère plusieurs éléments à partir d'un autre vecteur\n";
v_entiers v_ent2(2, 0);
cout << "Voici le vecteur cible v_ent2:\n";
AfficherVector(v_ent2);

cout << "v_ent2 après insertion depuis v_ent\n";
v_ent2.insert(v_ent2.begin() + 1, v_ent.begin() + 1, v_ent.begin() +
3);
AfficherVector(v_ent2);

cout << "On ajoute un élément à la fin de v_ent2";
cout << "avec push_back()\n";
v_ent2.push_back(100);
AfficherVector(v_ent2);

cout << "On supprime un élément à partir de la fin";
cout << " de v_ent2 avec pop_back()\n";
v_ent2.pop_back();
AfficherVector(v_ent2);

cout << "On supprime v_ent2\n";
v_ent2.clear();
AfficherVector(v_ent2);
}

//Définition de AfficherVector
template<class T, class A>
void AfficherVector(const vector<T, A>& v)
{

```

```

    cout << "Taille= " << v.size() << "\tCapacité= " << v.capacity() <<
    "\n";

    // On parcourt le vecteur via l'opérateur []
    cout << "éléments:\t";
    for (typename vector<T, A>::size_type i = 0; i < v.size(); ++i)
        cout << v[i] << ", ";

    cout << "\n\n";
}

```

Code 8.5 : opérations sur des vectors

```

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

typedef vector<int>    v_entiers;

//prototype de la fonction d'affichage
template<class T, class A>
void AfficherVector(const vector<T, A>& v);
//Prototype de la fonction de comparaison
template<class T, class A>
void ComparerVectors(const vector<T, A>& v1, const vector<T, A>& v2);

int main()
{
    cout << "On définit un vecteur d'entiers avec 5 éléments\n";
    v_entiers v_ent1(5);

    cout<<"On affecte une valeur à chaque élément via []\n";
    for (vector<int>::size_type i = 0; i < v_ent1.size(); ++i) {
        v_ent1[i] = i * i;
    }
    AfficherVector(v_ent1);

    cout << "On définit v_ent2 en recopiant les éléments ";
    cout << "de v_ent1 \nVoici v_ent2:" << endl;
    v_entiers v_ent2 = v_ent1;
}

```

57/82

```

    AfficherVector(v_ent2);

    cout << "On compare v_ent et v_ent2" << endl;
    ComparerVectors(v_ent1, v_ent2);

    cout << "On ajoute un élément à la fin de v_ent2\n";
    v_ent2.push_back(100);
    AfficherVector(v_ent2);

    cout << "On compare v_ent et v_ent2 de nouveau" << endl;
    ComparerVectors(v_ent1, v_ent2);

    cout << "On échange v_ent et v_ent2 avec swap()" << endl;
    v_ent1.swap(v_ent2);
    cout << "v_ent1 devient:\n";
    AfficherVector(v_ent1);
    cout << "v_ent2:\n";
    AfficherVector(v_ent2);
    cout << "Dernière comparaison:" << endl;
    ComparerVectors(v_ent1, v_ent2);
}

//Définition de la fonction d'affichage
template<class T, class A>
void AfficherVector(const vector<T, A>& v)
{
    cout << "Taille= " << v.size() << "\tCapacité = " << v.capacity() <<
    "\n";

    // On affiche les éléments du vecteur
    cout << "éléments:\t";
    for (typename vector<T,A>::size_type i = 0; i < v.size(); ++i)
        cout << v[i] << ", ";
    cout << "\n\n";
}

//Définition de la fonction de comparaison
template<class T, class A>
void ComparerVectors(const vector<T, A>& v1, const vector<T, A>& v2)
{

```

```

if (v1 == v2)    {
    cout << "v1 == v2";
}
else if (v1 < v2)
{
    cout << "v1 < v2";
}
else
{
    cout << "v1 > v2";
}
cout << "\n\n";
}

```

Code 8.6 : couper-coller entre deux listes

```

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

typedef list<int> l_entiers;
typedef list<int>::iterator Iter_lEntiers;

//Prototype de la fonction d'affichage
template<class T, class A>
void AfficherListe(const list<T, A>& une_liste);

int main()
{
    // on définit une liste d'entiers avec 5 éléments
    l_entiers Liste_1(5);
    int j = 0;
    for (Iter_lEntiers i1 = Liste_1.begin(); i1 != Liste_1.end(); ++i1)
        *i1 = 5 * j++;
    cout << "Liste_1" << "\n";
    AfficherListe(Liste_1);

    //On définit une liste d'entiers avec 6 éléments
    l_entiers Liste_2(6);
    j = 0;

```

59/82

```

    for (Iter_lEntiers i2 = Liste_2.begin(); i2 != Liste_2.end(); ++i2)
        *i2 = 100 * j++;
    cout << "Liste_2" << "\n";
    AfficherListe(Liste_2);

    // On exécute le couper-coller
    cout << "Après le couper-coller:\n";
    Liste_1.splice(++Liste_1.begin(), Liste_2, ++(++Liste_2.begin()), --
Liste_2.end());
    cout << "Liste_1" << "\n";
    AfficherListe(Liste_1);
    cout << "Liste_2" << "\n";
    AfficherListe(Liste_2);
}

//Définition de la fonction d'affichage
template<class T, class A>
void AfficherListe(const list<T, A>& une_liste)
{
    cout << "Taille = " << une_liste.size() << endl;
    cout << "Eléments :";
    for (typename list<T, A>::const_iterator i = une_liste.begin();
        i != une_liste.end(); ++i)
        cout << *i << ", ";
    cout << "\n\n";
}

```

Code 8.7 : ajout et suppression du premier élément d'une liste

```

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

typedef list<int>      l_entiers;
typedef list<int>::iterator  Iter_lEntiers;

template<class T, class A>
void AfficherListe(const list<T, A>& une_liste);

int main()

```

```

{
    // on définit une liste d'entiers avec 5 éléments
    l_entiers Liste_1(5);
    int j = 0;
    for (Iter_lEntiers ia = Liste_1.begin(); ia != Liste_1.end(); ++ia)
        *ia = 5 * j++;
    cout << "Liste_1" << "\n";
    AfficherListe(Liste_1);

    // On supprime le premier élément
    Liste_1.pop_front();
    cout << "On supprime le premier élément:\n";
    AfficherListe(Liste_1);

    // on insère un nouvel élément
    Liste_1.push_front(0);
    cout << "On ajoute 0 au début de la liste:\n";
    AfficherListe(Liste_1);
}

//Définition de la fonction d'affichage
template<class T, class A>
void AfficherListe(const list<T, A>& une_liste)
{
    cout << "Taille = " << une_liste.size() << endl;
    cout << "Eléments :";
    for (typename list<T, A>::const_iterator i = une_liste.begin();
         i != une_liste.end(); ++i)
        cout << *i << ", ";
    cout << "\n\n";
}

```

Code 8.8 : tri, fusion et inversement des éléments de liste

```

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

typedef list<int> l_entiers;
typedef list<int>::iterator Iter_lEntiers;

```

```

template<class T, class A>
void AfficherListe(const list<T, A>& une_liste);

int main()
{
    // on définit une liste d'entiers avec 5 éléments
    l_entiers Liste_1(5);
    int j = 0;
    for (Iter_lEntiers ia = Liste_1.begin(); ia != Liste_1.end(); ++ia)
        *ia = 5 * j++;
    cout << "Liste_1 :" << "\n";
    AfficherListe(Liste_1);

    // on définit une liste d'entiers avec 6 éléments
    l_entiers Liste_2(6);
    j = 10;
    for (Iter_lEntiers ib = Liste_2.begin(); ib != Liste_2.end(); ++ib)
        *ib = 2 * j--;
    cout << "Liste_2 :" << "\n";
    AfficherListe(Liste_2);

    //On crée 2 autres listes à partir des 2 premières
    l_entiers Liste_3 = Liste_1;
    l_entiers Liste_4 = Liste_2;

    //On fusionne les 2 premières
    cout << "Fusion des listes non triées :\n";
    Liste_1.merge(Liste_2);

    cout << "Liste_1" << "\n";
    AfficherListe(Liste_1);
    cout << "Liste_2" << "\n";
    AfficherListe(Liste_2);

    cout << "On inverse les éléments de Liste_1\n";
    Liste_1.reverse();
    AfficherListe(Liste_1);

    cout << "On trie Liste_3 et Liste_4 puis on fusionne:\n";

```

```

Liste_3.sort();
Liste_4.sort();
Liste_3.merge(Liste_4);

cout << "Liste_3" << "\n";
AfficherListe(Liste_3);
cout << "Liste_4" << "\n";
AfficherListe(Liste_4);
}

//Définition de la fonction d'affichage
template<class T, class A>
void AfficherListe(const list<T, A>& une_liste)
{
    cout << "Taille= " << une_liste.size() << endl;
    cout << "Eléments:";
    for (typename list<T, A>::const_iterator i = une_liste.begin();
         i != une_liste.end(); ++i)
        cout << *i << ", ";
    cout << "\n\n";
}

```

Code 8.9 : suppression d'éléments dans une liste

```

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

typedef list<int>    l_entiers;
typedef list<int>::iterator    Iter_lEntiers;

template<class T, class A>
void AfficherListe(const list<T, A>& une_liste);

int main()
{
    // on définit une liste d'entiers avec 5 éléments
    l_entiers Liste_1(5);
    int j = 0;
    for (Iter_lEntiers ia = Liste_1.begin(); ia != Liste_1.end(); ++ia)

```

```

        *ia = 5 * j++;
        cout << "Liste_1" << "\n";
        AfficherListe(Liste_1);

        cout << "On supprime le 5 dans Liste_1:\n";
        Liste_1.remove(5);
        AfficherListe(Liste_1);

        // on définit une liste d'entiers avec 6 éléments
        l_entiers Liste_2(6);
        j = 10;
        for (Iter_lEntiers ib = Liste_2.begin(); ib != Liste_2.end(); ++ib)
            *ib = 2 * j--;
        cout << "Liste_2" << "\n";
        AfficherListe(Liste_2);

        cout << "on colle Liste_1 et Liste_2";
        cout << " et on ajoute 10 au début de Liste_1:\n";
        Liste_1.splice(++(++Liste_1.begin()), Liste_2);
        Liste_1.push_front(10);

        cout << "Liste_1" << "\n";
        AfficherListe(Liste_1);

        cout << "On trie Liste_1:\n";
        Liste_1.sort();
        AfficherListe(Liste_1);

        cout<<"Maintenant, on supprime les doublons de Liste_1:";
        cout << "\n";
        Liste_1.unique();
        AfficherListe(Liste_1);
    }

    //
    // On affiche les éléments de la liste
    //
    template<class T, class A>
    void AfficherListe(const list<T, A>& une_liste)
    {

```

```

cout << "Taille= " << une_liste.size() << endl;
cout << "Eléments:";
for (typename list<T, A>::const_iterator i = une_liste.begin();
     i != une_liste.end(); ++i)
    cout << *i << ", ";
cout << "\n\n";
}

```

Code 8.10 : utilisation d'une pile

```

#include <iostream>
#include <stack>

using namespace std;

int main()
{
    typedef stack<int> pile_entiers;
    //On crée une pile :
    pile_entiers ma_pile;
    //On empile quelques éléments :
    ma_pile.push(1);
    ma_pile.push(2);
    ma_pile.push(3);
    //On affiche les éléments en ordre inverse :
    while (!ma_pile.empty()) //tant que la pile n'est pas vide
    {
        cout << ma_pile.top() << endl; //on affiche l'élément
        ma_pile.pop(); //puis on le supprime
    }
}

```

Code 8.11 : utilisation d'une file

```

#include <iostream>
#include <queue>

using namespace std;

int main()

```

```

{
    typedef queue<int> file;
    //On crée une file d'attente:
    file ma_file;
    //On ajoute quelques éléments :
    ma_file.push(1);
    ma_file.push(2);
    ma_file.push(3);

    while (!ma_file.empty()) //tant que la file n'est pas vide
    {
        cout << ma_file.front() << endl; //on affiche l'élément de tête
        ma_file.pop(); //puis on le supprime
    }
}

```

Code 8.12 : utilisation d'une file d'attente à priorité

```

#include <iostream>
#include <queue>

using namespace std;

// Type des objets stockés dans la file :
class A
{
public:
    int p; // Priorité
    const char *t; // Valeur chaîne de caractères

    A() : p(0), t(0) {}
    A(int p, const char *t) : p(p), t(t) {}

    // fonction de comparaison :
    bool operator()(const A &a1, const A &a2)
    {
        //On renvoie 0 ou 1 selon que la comparaison est
        //vraie ou fausse
        return a1.p < a2.p ;
    }
}

```

66/82

```

};

int main(void)
{
    //On crée quelques objets :
    A a1(1, "Priorité faible");
    A a2(2, "Priorité moyenne 1");
    A a3(2, "Priorité moyenne 2");
    A a4(3, "Forte priorité");
    //On crée la file d'attente à priorité :
    priority_queue<A, vector<A>, A> ma_file; /*attention, il faut insérer
    un espace entre les 2 > > pour ne pas confondre avec
    l'opérateur >> de cin*/
    //On ajoute les éléments :
    ma_file.push(a3);
    ma_file.push(a1);
    ma_file.push(a2);
    ma_file.push(a4);
    //On récupère les éléments par ordre de priorité :
    while (!ma_file.empty()) //tant que la file n'est pas vide
    {
        //on affiche l'élément de plus forte priorité :
        cout << ma_file.top().t << endl;
        ma_file.pop(); //puis on le supprime
    }
}

```

Code 8.13 : utilisation d'un map

```

#include <iostream>
#include <string>
#include <map>
using namespace std;

class MaClasse
{
public:
    //constructeurs
    MaClasse():v1("Objet nouveau"), v2(0) {}
}

```

67/82

```

    MaClasse(string valeur1, int valeur2 = 0):
        v1(valeur1), v2(valeur2) {}

//Fonctions d'accès
void    DefinirValeur(string valeur1) { v1 = valeur1; }
void    DefinirValeur(int valeur2) { v2 = valeur2; }
string  LireValeur() const { return v1; }
int     LireValeur2() const { return v2; }

//opérateur << surchargé
friend ostream& operator<<(ostream& sortie, const MaClasse& p)
{
    sortie << "Chaîne: " << p.v1
        << "\tValeur: " << p.v2;
    return sortie;
}

private:
    string v1;
    int v2;
};

// Fonction modèle d'affichage des éléments d'un map
template<class T, class A>
void AfficherMap(const map<T, A>& m);

int main()
{
    //On crée 3 objets
    MaClasse objet1("Objet 1", 10);
    MaClasse objet2("Objet 2", 20);
    MaClasse objet3("Objet 3", 30);

    //On crée le map
    map<string, MaClasse> MonMap;
    //On initialise les éléments
    MonMap[objet1.LireValeur()] = objet1;
    MonMap[objet2.LireValeur()] = objet2;
    MonMap[objet3.LireValeur()] = objet3;
}

```

```

AfficherMap(MonMap);

//On affiche la valeur de objet1
cout << "Voici les valeurs de objet1: " << MonMap["Objet
1"].LireValeur();
cout << "\t" << MonMap["Objet 1"].LireValeur2() << endl;

//On modifie les 2 valeurs de objet2
MonMap["Objet 2"].DefinirValeur("Objet 2 bis");
MonMap["Objet 2"].DefinirValeur(22);
cout << "Voici les nouvelles valeurs de objet2: ";
cout << MonMap["Objet 2"].LireValeur() << "\t";
cout << MonMap["Objet 2"].LireValeur2() << "\n";

cout << "On ajoute au map : Chaîne: Objet 1 Valeur: 40\n";
MaClasse objet4("Objet 1", 40);
MonMap[objet4.LireValeur()] = objet4;
AfficherMap(MonMap);

// Les diverses fonctions d'accès relatives à l'objet 1
cout << "\nVoici le nombre d'objets 1 : ";
cout << MonMap.count("Objet 1") << "\n";

//On cherche le premier objet de clé "Objet 1"
map<string, MaClasse>::iterator ci=MonMap.lower_bound("Objet 1");
cout << "Le premier Objet 1: " << ci->second << "\n";

//on pointe sur l'élément suivant
ci = MonMap.upper_bound("Objet 1");
cout << "Élément qui suit Objet 1: " << ci->second;

// On tente d'accéder à un élément qui n'existe pas
cout << "\nOn cherche un Objet 10 : ";
cout << MonMap["Objet 10"] << endl;

cout << "On efface Objet 3\n";
MonMap.erase("Objet 3");
AfficherMap(MonMap);

cout << "On efface tout\n";

```

```

MonMap.clear();
    AfficherMap(MonMap);

}

//Définition de la fonction d'affichage du map
template<class T, class A>
void AfficherMap(const map<T, A>& m)
{
    cout << "Voici les éléments du map :\n";
    for (typename map<T, A>::const_iterator ci = m.begin(); ci != m.end();
        ++ci)
        cout << ci->first << "\t" << ci->second << "\n";

    cout << "\n\n";
}

```

Code 8.14 : utilisation de la classe *multimap*

```

#include <iostream>
#include <string>
#include <map>
using namespace std;

template<class Cle, class T>
void AfficherMultimap(const multimap<Cle, T>& m);

template<class Cle, class T>
void AfficherMultimapRange(const multimap<Cle, T>& m, const Cle& k);

int main()
{
    //On crée un multimap
    multimap<string, string> MonMultimap;
    //On y stocke des éléments
    MonMultimap.insert(make_pair((string)"cle_1", (string)"Elément 1"));
    MonMultimap.insert(make_pair((string)"cle_1", (string)"Elément 2"));
    MonMultimap.insert(make_pair((string)"cle_1", (string)"Elément 3"));
    MonMultimap.insert(make_pair((string)"cle_2", (string)"Elément 4"));
    MonMultimap.insert(make_pair((string)"cle_3", (string)"Elément 5"));
}

```

70/82

```

cout << "Voici le contenu de notre multimap:\n";
AfficherMultimap(MonMultimap);
cout << "Voici les éléments de valeur de";
cout << " clé \"cle_1\":\n";
AfficherMultimapRange(MonMultimap, (string)"cle_1");
}

template<class Cle, class T>
void AfficherMultimap(const multimap<Cle, T>& m)
{
    typedef typename multimap<Cle, T>::const_iterator Itor;

    for (Itor i = m.begin(); i != m.end(); ++i)
        cout << i->first << "\t" << i->second << "\n";
    cout << "\n";
}

template<class Cle, class T>
void AfficherMultimapRange(const multimap<Cle, T>& m, const Cle& k)
{
    typedef typename multimap<Cle, T>::const_iterator Itor;

    pair<Itor, Itor> p = m.equal_range(k);
    for (Itor i = p.first; i != p.second; ++i)
        cout << i->first << "\t" << i->second << "\n";
    cout << "\n";
}

```

Code 9.1 : algorithme de génération et de stockage d'éléments dans un conteneur

```

#include <iostream>
#include <list>
#include <iterator>
#include <algorithm>

using namespace std;

//Fonction de génération des valeurs à stocker

```

```

int Generateur()
{
    static int i = 0;
    return i++;
}

int main()
{
    typedef list<int> li; //li est le type liste d'entiers
    li ma_liste;          //on crée une liste
    //On l'alimente avec les 20 premiers entiers :
    generate_n(back_inserter(ma_liste), 20, Generateur);
    //On affiche la liste :
    li::iterator i = ma_liste.begin();
    while (i != ma_liste.end())
    {
        cout << *i << endl;
        i++;
    }
}

```

Code 9.2 : algorithmes de copie

```

#include <iostream>
#include <algorithm>
#include <cstring> //pour la fonction strlen()

using namespace std;

int main()
{
    char chaine[] = "abcdefg123";
    cout << "Voici la chaine initiale: " << chaine << endl;
    //On repère la fin de la chaîne en mémoire:
    //char *p_debut = chaine;
    char *p_fin = chaine + strlen(chaine);
    //On recopie les 3 derniers caractères au début de la chaîne
    copy(p_fin-3, p_fin, chaine);
    //On affiche le résultat :
    cout <<"On recopie les 3 derniers caractères au début de la chaîne: ";
}

```

72/82

```

    cout << chaine << endl;
    //On écrase la chaîne par elle-même à partir du 4ème élément :
    copy_backward(chaine, p_fin-3, p_fin);
    // On affiche le résultat :
    cout << "On écrase la chaîne par elle-même à partir du 4ème élément: "
    cout <<chaine << endl;
}

```

Code 9.3 : algorithme d'échange d'éléments

```

#include <iostream>
#include <list>
#include <algorithm>

using namespace std;

typedef list<int> li;
void AfficherListe(li);

int main()
{
    //On crée une liste d'entiers :
    li ma_liste;
    //On la remplit
    ma_liste.push_back(1);
    ma_liste.push_back(2);
    ma_liste.push_back(3);
    ma_liste.push_back(4);

    cout << "Voici la liste initiale: \n";
    AfficherListe(ma_liste);

    // On crée un tableau de 4 éléments initialisés:
    int tab[4] = {10, 20, 30, 40};

    cout << "Voici le tableau initial: \n";
    for (int i=0; i<4; i++)
        cout << tab[i] << " ";
    cout << endl;
}

```

```

//On échange le contenu des 2 conteneurs :
swap_ranges(tab, tab+4, ma_liste.begin());

cout << "Voici la nouvelle liste: \n";
AfficherListe(ma_liste);

cout << "Voici le nouveau tableau: \n";
for (int j=0; j<4; j++)
    cout << tab[j] << " ";
    cout << endl;

}

//Définition de l'affichage de la liste
void AfficherListe(li l)
{
    li::iterator iter = l.begin();
    while (iter != l.end())
    {
        cout << *iter << " ";
        iter++;
    }
    cout << endl;
}

```

Code 9.4 : algorithme de recherche et de remplacement

```

#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    int tab[10] = {1, 2, 10, 9, 3, 2, 9, 4, 2, 9};
    //On remplace tous les 9 par des 3 :
    replace(tab, tab+10, 9, 3);
    //On affiche le résultat :
    int i;
    for (i=0; i<10; i++)

```

```
    cout << tab[i] << endl;
}
```

Code 9.5 : algorithme de rotation

```
include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    int tab[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    cout << "Voici le tableau initial: \n";
    for (int i=0; i<10; i++)
        cout << tab[i] << " ";
    cout << endl;

    //On effectue une rotation pour amener le quatrième
    // élément en première position :
    rotate(tab, tab+3, tab+10);
    cout << "Voici le tableau après rotation: \n";
    for (int i=0; i<10; i++)
        cout << tab[i] << " ";
    cout << endl;
}
```

Code 9.6 : algorithme de permutation

```
#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    int tab[3] = {1, 1, 2};
    //On affiche l'ensemble des permutations de (1, 1, 2) :
    do
    {
```

```
    int i;
    for (i=0; i<3; i++)
        cout << tab[i] << " ";
    cout << endl;
}
while (next_permutation(tab, tab+3));
}
```

Code 9.7 : algorithme d'inversion

```
#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    int tab[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    //On inverse le tableau :
    reverse(tab, tab+10);
    //On affiche le résultat :
    int i;
    for (i=0; i<10; i++)
        cout << tab[i] << endl;
}
```

Code 9.8 : algorithme de réorganisation aléatoire

```
#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    int tab[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    //On inverse le tableau :
    reverse(tab, tab+10);
    //On affiche le résultat :
    int i;
```

```
    for (i=0; i<10; i++)
        cout << tab[i] << endl;
}
```

Code 9.9 : algorithmes d'itération

```
#include <iostream>
#include <functional>
#include <algorithm>

using namespace std;

void aff_entier(int i)
{
    cout << i << "\t";
}

int main()
{
    int tab[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    //On inverse tous les éléments du tableau :
    transform(tab, tab+10, tab, negate<int>());
    //On affiche le résultat :
    for_each(tab, tab+10, ptr_fun(&aff_entier));
}
```

Code 9.10 : algorithme de décompte d'éléments

```
include <iostream>
#include <functional>
#include <algorithm>

using namespace std;

bool parity_even(int i)
{
    return (i & 1) == 0;
}

int main()
```

77/82

```

{
    int tab[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    //On compte le nombre d'éléments pairs :
    cout << "Nombre d'éléments pairs dans le tableau
{0,1,2,3,4,5,6,7,8,9}: ";
    cout << count_if(tab, tab+10, ptr_fun(&parity_even)) << endl;
}

```

Code 9.11 : algorithme de recherche d'éléments

```

#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    int tab[10] = {0, 5, 3, 4, 255, 7, 0, 5, 255, 9};
    //On recherche les éléments de valeur 0 ou 255 :
    int sep[2] = {0, 255};
    int *debut = tab;
    int *fin = tab+10;
    int *courant;
    cout << "Dans le tableau {0,5,3,4,255,7,0,5,255,9}" << endl;
    while ((courant=find_first_of(debut, fin, sep, sep+2)) != fin)
    {
        //On affiche la position de l'élément trouvé :
        cout << *courant << " en position " << courant-tab << endl;
        debut = courant+1;
    }
}

```

Code 9.12 : algorithmes de recherche de séquence spécifique

```

#include <iostream>
#include <algorithm>

using namespace std;

int main()

```

```

{
    int tab[10] = {1, 2, 4, 5, 3, 1, 2, 3, 5, 9};
    //On recherche le schéma {1, 2, 3} dans le tableau :
    int motif[3] = {1, 2, 3};
    int *p = search(tab, tab+10, motif, motif+3);
    cout << "Dans le tableau {1,2,4,5,3,1,2,3,5,9}" << endl;
    cout << "{1,2,3} se trouve en position ";
    cout << p - tab << endl;
    //On recherche la dernière occurrence de {1, 2} :
    p = find_end(tab, tab+10, motif, motif+2);
    cout << "La dernière occurrence de {1, 2}";
    cout << " se trouve en position " << p - tab << endl;
}

```

Code 9.13 : algorithme de recherche de doublons

```

#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    int tab[10] = {0, 1, 2, 2, 3, 4, 4, 5, 6, 2};
    // Recherche les doublons dans le tableau :
    int *debut = tab;
    int *fin = tab+10;
    int *p;
    cout << "Dans le tableau {0, 1, 2, 2, 3, 4, 4, 5, 6, 2}" << endl;
    while ((p = adjacent_find(debut, fin)) != fin)
    {
        cout << "Doublon en position " << p-tab << endl;
        debut = p+1;
    }
}

```

Code 9.14 : algorithme de tri

```

#include <iostream>
#include <algorithm>

```

```

using namespace std;

int main()
{
    int tab[10] = {2, 3, 7, 5, 4, 1, 8, 0, 9, 6};
    cout << "On trie le tableau {2, 3, 7, 5, 4, 1, 8, 0, 9, 6} :" << endl;
    sort(tab, tab+10);
    //On affiche le résultat :
    int i;
    for (i=0; i<10; i++)
        cout << tab[i] << " ";
    cout << endl;
}

```

Code 9.15 : algorithme de tri partiel

```

#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    int tab[10] = {2, 3, 7, 5, 4, 1, 8, 0, 9, 6};
    cout << "Le tableau {2, 3, 7, 5, 4, 1, 8, 0, 9, 6} devient :" << endl;
    // Trie les 5 premiers éléments du tableau :
    partial_sort(tab, tab+5, tab+10);
    //On affiche le résultat :
    int i;
    for (i=0; i<10; i++)
        cout << tab[i] << " ";
    cout << endl;
}

```

Code 9.16 : algorithme de positionnement du n-ième élément

```

include <iostream>
#include <algorithm>

```

```

using namespace std;

int main()
{
    int tab[10] = {2, 3, 9, 6, 7, 5, 4, 0, 1, 8};
    cout << "Tableau initial : {2, 3, 9, 6, 7, 5, 4, 0, 1, 8}" << endl;
    // Trie tous les éléments un à un :
    int i;
    for (i=0; i<10; i++)
    {
        nth_element(tab, tab+i, tab+10);
        cout << "L'élément " << i <<
            " a pour valeur " << tab[i] << endl;
    }
}

```

Code 9.17 : algorithmes de détermination du maximum et du minimum

```

include <iostream>
#include <algorithm>

using namespace std;

int main()
{
    int tab[10] = {5, 2, 4, 6, 3, 7, 9, 1, 0, 8};
    cout << "Dans le tableau : {5, 2, 4, 6, 3, 7, 9, 1, 0, 8}" << endl;
    //On affiche le minimum et le maximum :
    cout << "Le minimum est: ";
    cout << *min_element(tab, tab+10) << endl;
    cout << "Le maximum est: ";
    cout << *max_element(tab, tab+10) << endl;
}

```

Code 10.1 : Hello World

```

1: #using <mcorlib.dll>
2:
3: using namespace System;

```

81/82

```
4:
5:int main()
6:{
7:  Console::WriteLine(L"Hello World via les extensions managées pour
  C++!");
8:}
```