INF 2005 Programmation orientée objet avec C++

Module 4 - Solutions

1.

```
#include <iostream>
#include <time.h>
using namespace std;
class Temps {
  public:
  Temps();
  void setHeure(int h) { heure = (h >= 0 && h < 24) ? h : 0; }
  void setMinute(int m) { minute = (m >= 0 && m < 60) ? m : 0; }
  void setSecond(int s) { second = (s >= 0 && s < 60) ? s : 0; }
  int getHeure(void) { return heure; }
  int getMinute(void) { return minute; }
  int getSecond(void) { return second; }
  void AfficherStandard(void);
  private:
    int heure;
    int minute;
    int second;
};

// définition des fonctions membres de la classe Temps

Temps::Temps()
{

  long int tempsTotal;     // Temps en secondes
                           // depuis 1970
  int AnneeCourante = 1994 - 1970;        // Temps écoulé jusqu'à
                                          // l'année courante
  double anneeTotale;      // Année courante
  double jourTotal;        // Nombre de jours écoulés
                           //depuis le début de l'année
  double jour;     // Jour courant
  long double facteur;      // Facteur de conversion
  int DecalageHoraire = 7;         // Le temps retourné par
                                   // la classe Temps() est
```

```
                                    // donnée par le nombre
                                   // de secondes écoulées
                                   // depuis 1/1/70 GMT. Selon
                                   //la zone, on doit décaler le
                                   //temps d'un certain nombre
                                   // horaire est de 7 heures.
  tempsTotal = time(NULL);
  facteur = (60.0 * 60.0 * 24.0 * 365.0); // Nombre total de
                                          // secondes dans 1 an.
  anneeTotale = tempsTotal / facteur - AnneeCourante;
  jourTotal = 365 * anneeTotale;  // Les années bissextiles sont
                                  // ignorées.
  jour = jourTotal - (int) jourTotal;

  setHeure(jour * 24 + DecalageHoraire);
  setMinute((jour * 24 - (int)(jour * 24)) * 60);
  setSecond((minute * 60 - (int)(minute * 60)) * 60);

}

void Temps::AfficherStandard()
{

  cout << ((heure % 12 == 0) ? 12 : heure % 12) << ":"
  << (minute < 10 ? "0" : "") << minute << ":"
  << (second < 10 ? "0" : "") << second
  << (heure < 12 ? " AM" : " PM");
}

// Programme principal

main()
{
  Temps t;
  t.AfficherStandard();
  return 0;
}
```

2.

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

class Rationnel {
  public:

    Rationnel(int = 0, int = 1); // Constructeur par défaut
    Rationnel addition(const Rationnel &);
    Rationnel soustraction(const Rationnel &);
    Rationnel multiplication(const Rationnel &);
```

```
    Rationnel division(Rationnel &);
  void AfficherRationnel(void);
  void AfficherRationnelVirgule(void);
  private:
    int numerateur;
    int denominateur;

    void simplification(void); // Fonction utilitaire (définie
                               // comme membre privé)

};

///#endif
// Définition des fonctions membres

Rationnel::Rationnel(int n, int d)
{
  numerateur = n;
  denominateur = d;
}
Rationnel Rationnel::addition(const Rationnel &a)
{
  Rationnel t;
  t.numerateur = a.numerateur * denominateur + a.denominateur *
numerateur;
  t.denominateur = a.denominateur * denominateur;
  t.simplification();
  return t;
}

Rationnel Rationnel::soustraction(const Rationnel &s)
{

  Rationnel t;
  t.numerateur = s.denominateur * numerateur - denominateur *
s.numerateur;
  t.denominateur = s.denominateur * denominateur;
  t.simplification();
}

Rationnel Rationnel::multiplication(const Rationnel &m)
{
  Rationnel t;
  t.numerateur = m.numerateur * numerateur;
  t.denominateur = m.denominateur * denominateur;
  t.simplification();
  return t;
}

Rationnel Rationnel::division(Rationnel &v)
```

```
{
  Rationnel t;
  t.numerateur = v.denominateur * numerateur;
  t.denominateur = denominateur * v.numerateur;
  t.simplification();
  return t;
 }

void Rationnel::AfficherRationnel(void)
{

  if (denominateur == 0)

    cout << "\nERREUR DE DIVISION PAR ZÉRO!!!" << endl;

  else if (numerateur == 0)
    cout << 0;
  else

    cout << numerateur << "/" << denominateur;
}

void Rationnel::AfficherRationnelVirgule(void)
{
  cout << (float) numerateur / denominateur;
}

void Rationnel::simplification(void)
{
  int superieur = numerateur > denominateur ? numerateur :
denominateur;
  int pgdc = 0; // Plus grand diviseur commun

  for (int boucle = 2; boucle <= superieur; boucle++)
    if (numerateur % boucle == 0 && denominateur % boucle == 0)
      pgdc = boucle;
    if (pgdc != 0)
    {
      numerateur /= pgdc;
      denominateur /= pgdc;
    }
}

// Programme principal

main()
{
  Rationnel c(1,3), d(7,8), x;

  c.AfficherRationnel();
```

```
  cout << " + ";
  d.AfficherRationnel();
  x = c.addition(d);
  cout << " = ";
  x.AfficherRationnel();
  cout << endl;
  x.AfficherRationnel();
  cout << " = ";
  x.AfficherRationnelVirgule();
  cout << endl << endl;

  c.AfficherRationnel();
  cout << " - ";
  d.AfficherRationnel();
  x = c.soustraction(d);
  cout << " = ";
  x.AfficherRationnel();
  cout << endl;
  x.AfficherRationnel();
  cout << " = ";
  x.AfficherRationnelVirgule();
  cout << endl << endl;

  c.AfficherRationnel();
  cout << " x ";
  d.AfficherRationnel();
  x = c.multiplication(d);
  cout << " = ";
  x.AfficherRationnel();
  cout << endl;
  x.AfficherRationnel();
  cout << " = ";
  x.AfficherRationnelVirgule();

  cout << endl << endl;
  c.AfficherRationnel();
  cout << " / ";
  d.AfficherRationnel();
  x = c.division(d);
  cout << " = ";
  x.AfficherRationnel();
  cout << endl;
  x.AfficherRationnel();
  cout << " = ";
  x.AfficherRationnelVirgule();
  cout << endl;

  return 0;
}
```

3.

```
#include <iostream>
#include <iomanip>
#include <string.h>
using namespace std;

class Reference
{
  public: char titre[256];
  char auteur[64];
  float prix;
 char type_argent[4];
  Reference(const char *btitre, const char *bauteur, const char
*bediteur, float bprix, const char *bdevise);
  void afficher_titre(void) { cout << titre << '\n'; }
  float donner_prix(void) { return(prix); }
  void afficher_livre(void)
  {
  afficher_titre();
  afficher_editeur();
  }
  void affecter_editeur(const char *nom) { strcpy(editeur, nom); }
  private:
  char editeur[256];
  void afficher_editeur(void) { cout << editeur << '\n'; }
};

Reference::Reference(const char *btitre, const  char *bauteur,
const char *bediteur, float bprix, const char *bdevise)
{
  strcpy(titre, btitre);
  strcpy(auteur, bauteur);
  strcpy(editeur, bediteur);
  prix = bprix;
  strcpy(type_argent, bdevise);
}

int main(void)
{
  Reference astuces("J'aime programmer en C/C++", "Jacques et
Rénald", "TELE-UNIVERSITE", 19, "$$");
  Reference genial("Programmation objet", "Samuel Pierre", "TELE-
UNIVERSITE",19, "$$");
  astuces.afficher_livre();
  genial.afficher_livre();
}
```

4.

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

class objet
{
public:

        objet(void);
void afficher_objet(void);
private:
int valeur1;
int valeur2;
int valeur3;
};

objet::objet(void) : valeur1(50), valeur2(500), valeur3(5000) { };
// Constructeur.
void objet::afficher_objet(void)
{
     cout << "Valeur 1 contient : " << valeur1 << endl;
cout << "Valeur 2 contient : " << valeur2 << endl;
cout << "Valeur 3 contient : " << valeur3 << endl;
}

int main(void)
{
     objet nombres;
nombres.afficher_objet();
}
```

5.

```
// Fichier M4_Ex5.h
///#ifndef M4_Ex5_H
///#define M4_Ex5_H
#include <iostream>
#include <iomanip>
using namespace std;
class CompteEpargne {
public:
     CompteEpargne(double b) { Balance= b >= 0 ? b : 0; }
void CalculerIntMensuel(void);
static void CalculerIntMensuel(double);
void AfficherBalance(void);
private:
     double Balance;
static double TauxIntAn;
```

```
};
///#endif // Fichier Ch5_Ex5.cpp
// Définition des fonctions membres
///#include "Ch5_Ex5.h"
///#include <iostream>
///#include <iomanip> // Initialisation des membres statiques
double CompteEpargne::TauxIntAn = 0.0; void
CompteEpargne::CalculerIntMensuel(void) { Balance+= Balance*
(TauxIntAn / 12.0);
} void CompteEpargne::CalculerIntMensuel(double i)
     { TauxIntAn = (i >= 0 && i <= 1.0) ? i : .03; }
void CompteEpargne::AfficherBalance(void)
{ cout.setf(ios::fixed | ios::showpoint);
cout << '$' << setprecision(2) << Balance;
} // Programme principal de Ch5_Ex5.cpp
///#include <iostream>
///#include <iomanip>
///using namespace std;
///#include " Ch5_Ex5.h"
main()
{ CompteEpargne epargne1(2000.0), epargne2(3000.0);
CompteEpargne::CalculerIntMensuel(.03); cout << endl << "Affichage
des balances mensuelles pour un an à un taux de 3 %" << endl <<
"Balance: Compte 1 "; epargne1.AfficherBalance(); cout <<
"\tCompte 2"; epargne2.AfficherBalance(); for (int mois = 1; mois
<= 12; mois++) { epargne1.CalculerIntMensuel();
epargne2.CalculerIntMensuel();
cout << endl << "Mois" << setw(3) << mois << ": Compte 1 ";
epargne1.AfficherBalance();
cout << "\tCompte 2 ";
epargne2.AfficherBalance();
}
CompteEpargne::CalculerIntMensuel(.04);
epargne1.CalculerIntMensuel();
epargne2.CalculerIntMensuel();
cout << endl << "Après avoir fixé le taux d'intérêt à 4 %" << endl
<< "Balances: Compte 1 ";
epargne1.AfficherBalance();
cout << "\tCompte 2 ";
epargne2.AfficherBalance();
return 0;
}
```

6.

```
// Fichier M4_Ex6.H
///#ifndef M4_Ex6.H
///#define M4_Ex6.H
#include <iostream>
#include <iomanip>
```

```
using namespace std; class Polynome { public:
     Polynome();
Polynome operator+(const Polynome&) const;
Polynome operator-(const Polynome&) const;
Polynome operator*(const Polynome&);
Polynome& operator+=(const Polynome&);
Polynome& operator-=(const Polynome&);
Polynome& operator*=(const Polynome&);
void entrerTermes(void);
void AfficherPolynome(void) const;
private:
     int exposants[100];
int coefficients[100];
void CombinerPolynome(Polynome&); // Combiner les termes communs.
};
///#endif
// Fichier M4_Ex6.cpp
// Définition des fonctions membres
///#include <iomanip.h>
///#include "Ch4_Ex3.h"
Polynome::Polynome()
{
     for (int t = 0; t < 100; t++) {
          coefficients[t] = 0;
exposants[t] = 0;
     }
}
void Polynome::AfficherPolynome(void) const
{ int debut, zero = 0; if (coefficients[0]) { // Affichage de la
constante
          cout << coefficients[0]; debut = 1; zero = 1; // Le
polynôme contient au moins un terme.
     }
else {          if (coefficients[1]) {
           cout << coefficients[1] << 'x'; // Le terme constant du
polymôme
// n'existe pas.
          if ((exposants[1] != 0) && (exposants[1] != 1))
               cout << '^' << exposants[1];
          zero = 1; // Le polynôme contient au moins un terme.
}
debut = 2;
} // Affichage des termes restants du polynôme
for (int x = debut; x < 100; x++) {
          if (coefficients[x] != 0) {                cout <<
setiosflags(ios::showpos) << coefficients[x] <<
resetiosflags(ios::showpos) << 'x'; if ((exposants[x] != 0) &&
(exposants[x] != 1))                       cout << '^' <<
exposants[x];
```

```
                zero = 1; // Le polynôme contient au moins un
terme.
           }
}
if (zero == 0) // Le polynôme ne contient aucun terme.

           cout << '0';
     cout << endl;
}
Polynome Polynome::operator+(const Polynome& r) const
{
Polynome temp;
int s;
int x;
int exposantExiste;
// Traiter l'élément dont l'exposant est nul.
temp.coefficients[0] = coefficients[0] + r.coefficients[0];
// Mettre les coefficients et exposants des polynômes
// dans des variables temporaires.
for ( s = 1; (s < 100) && (r.exposants[s] != 0); s++) {
           temp.coefficients[s] = r.coefficients[s];
temp.exposants[s] = r.exposants[s]; }
for ( x = 1; x < 100; x++) {
   exposantExiste = 0; // En supposant que l'exposant n'existe
pas.
for (int t = 1; (t < 100) && (exposantExiste == 0); t++)
   if (exposants[x] == temp.exposants[t]) {
      temp.coefficients[t] += coefficients[x];
exposantExiste = 1; // Exposant trouvé.
} // Si l'exposant n'existe pas, l'insérer dans temp.
if (exposantExiste == 0) {
   temp.exposants[s] = exposants[x];
temp.coefficients[s] += coefficients[x];
++s; } }
return temp;
}
Polynome &Polynome::operator+=(const Polynome &r)
{
     *this = *this + r;
return *this;
}
Polynome Polynome::operator-(const Polynome& r) const
{
Polynome temp;
int exposantExiste;
int s, x;
// Traiter l'élément dont l'exposant est nul.
temp.coefficients[0] = coefficients[0] - r.coefficients[0];
// Mettre les coefficients et exposants des polynômes
// dans des variables temporaires.
```

```
for ( s = 1; (s < 100) && (exposants[s] != 0); s++) {
          temp.coefficients[s] = coefficients[s];
temp.exposants[s] = exposants[s]; }
for ( x = 1; x < 100; x++) {
exposantExiste = 0; // En supposant que l'exposant n'existe pas.
for (int t = 1; (t < 100) && (exposantExiste == 0); t++)
if (r.exposants[x] == temp.exposants[t]) {
     temp.coefficients[t] -= r.coefficients[x];
exposantExiste = 1; // Exposant trouvé.
} // Si l'exposant n'existe pas, l'insérer dans temp.
if (exposantExiste == 0) {
     temp.exposants[s] = r.exposants[x];
temp.coefficients[s] -= r.coefficients[x]; ++s; }
   } return temp; }
Polynome &Polynome::operator-=(const Polynome& r)
{
     *this = *this - r;
return *this;
} Polynome Polynome::operator*(const Polynome& r)
{
Polynome temp;
int s = 1;
for (int x = 0; (x < 100) && (x == 0 || coefficients[x] != 0);
x++)
 for (int y = 0; (y < 100) && (y == 0 || r.coefficients[y] != 0);
y++)
      if (coefficients[x] * r.coefficients[y])
        if ((exposants[x] == 0) && (r.exposants[y] == 0))
          temp.coefficients[0] += coefficients[x] *
r.coefficients[y];
        else {
 temp.coefficients[s] = coefficients[x] * r.coefficients[y];
temp.exposants[s] = exposants[x] + r.exposants[y];
++s; }
 CombinerPolynome(temp); // Combiner les termes communs.
return temp;
}

void Polynome::CombinerPolynome(Polynome& w)
{
Polynome temp = w;
int exp,x;
for (x = 0; x < 100; x++) {      w.coefficients[x] = 0;
w.exposants[x] = 0;
}
for (x = 1; x < 100; x++) {
          exp = temp.exposants[x];
for (int y = x + 1; y < 100; y++)
if (exp == temp.exposants[y]) {
temp.coefficients[x] += temp.coefficients[y];
```

```
temp.exposants[y] = 0;
temp.coefficients[y] = 0; } }
w = temp;
}
Polynome &Polynome::operator*=(const Polynome& r)
{     *this = *this * r;         return *this;
}
void Polynome::entrerTermes(void)
{
int trouve = 0, nombreDeTermes, c, e;
int terme;
cout << endl << "Entrer le nombre de termes du polynôme : ";
cin >> nombreDeTermes; for (int n = 1; n <= nombreDeTermes; n++) {
cout << endl << "Entrer le coefficient : ";
cin >> c;
cout << "Entrer l'exposant : ";
cin >> e;
if (c != 0) {

 // L'élément d'exposant nul correspond au premier élément.
if (e == 0) {
 coefficients[0] += c;
 continue;
}
for (terme = 1; (terme < 100) && (coefficients[terme] != 0);
terme++)
if (e == exposants[terme]) {
coefficients[terme] += c; exposants[terme] = e; trouve = 1; //
Exposant existant est mis à jour.
}
if (trouve == 0) { // Ajouter des termes.
coefficients[terme] += c; exposants[terme] = e; }
     }
   }
}

// Programme principal ///#include "Ch4_Ex3.h"
main()
{ Polynome a, b, c, t;
a.entrerTermes();
b.entrerTermes();
cout << "Le premier polynôme est : " << endl;
a.AfficherPolynome();
cout << "Le second polynôme est : " << endl;
b.AfficherPolynome();
cout << endl << "L'addition des polynômes aboutit au résultat : "
<< endl;
c = a + b;
c.AfficherPolynome();
```

```
cout<<endl<< "L'opérateur += appliqué aux polynômes conduit au
résultat : " << endl;
t = a; // Sauvegarder la valeur de a.
a += b;
a.AfficherPolynome();
cout << endl << "La soustraction des polynômes aboutit au résultat
: " << endl;
a = t; // Réinitialiser a à sa valeur originale.
c = a - b;
c.AfficherPolynome();
cout << endl << "L'opérateur -= appliqué aux polynômes conduit au
résultat : " << endl;
a -= b;
a.AfficherPolynome();
cout << endl << "La multiplication des polynômes aboutit au
résultat : " << endl;
a = t; // Réinitialiser a à sa valeur originale.
c = a * b;
c.AfficherPolynome();
cout<<endl<< "L'opérateur *= appliqué aux polynômes conduit au
résultat : " << endl;
a *= b;
a.AfficherPolynome();
cout << endl;
return 0;
}
```

7.

```
#include <iostream>
#include <string.h>
using namespace std;
class Emission
{
public:
        char titre[64];
char premier_role[64];
char second_role[64];
void afficher_Emission(void);
void initialisation(const char *titre_Emission, const char
*premier, const char *second);

};

void Emission::afficher_Emission(void)
{

        cout << "Titre de l'émission : " << titre << endl;
cout << "Acteurs : " << premier_role << " et " << second_role <<
endl
```

```
<< endl;
}

void Emission::initialisation(const char *titre_Emission, const
char *premier, const char *second)
{

strcpy(titre, titre_Emission);
strcpy(premier_role, premier);
strcpy(second_role, second);
}
main()
{
Emission Omerta, Petite_Vie;

Omerta.initialisation("Omerta", "Romano Orzani", "Pierre
Gauthier");
Petite_Vie.initialisation("Petite_Vie", "Marc Messier", "Marc
Labrèche");
Omerta.afficher_Emission();
}
```

8.

```
#include <iostream>
using namespace std;
class simple
{
        int ami1, ami2;
public:
        friend int somme(simple x);
void init_ab(int i, int j);
};
void simple::init_ab(int i, int j)
{
ami1 = i;
ami2 = j;
}
int somme(simple objet)
{
        // Parce que somme est une amie de simple, elle peut
accéder
// directement à ami1 et ami2.
return objet.ami1 + objet.ami2;

}
int main(void)
{
simple entier;
```

```
cout << "La somme des nombres 777 et 999 vaut :" << endl;
entier.init_ab(777,999);
cout << somme(entier) << endl;
}
```

9.

```
#include <iostream>
using namespace std;
class conversion
{
protected:
double val1;
double val2;
public:
conversion(double i) { val1 = i; }
double recuperer_conversion(void) {return val2;}
double recuperer_init(void) {return val1;}
virtual void calculer(void) = 0;
};
// Litres en gallons
class Nombre_litre : public conversion
{
 public:
 Nombre_litre(double i) : conversion(i) { }
void calculer(void) { val2 = val1 / 3.7854; }
};
// Fahrenheit en Celsius
class Temperature_Fahrenheit : public conversion
{ public:
      Temperature_Fahrenheit (double i) : conversion(i) { }
void calculer(void) { val2 = (val1 - 32) / 1.8; }
};
int main(void)
{   conversion *p; // Pointeur sur classe de base
Nombre_litre lgob(40);
Temperature_Fahrenheit fcob(75);
p = &lgob; // Convertit litres en gallons.
cout << p->recuperer_init() << " litres ";
p->calculer();
cout << p->recuperer_conversion() << " gallons." << endl;
p = &fcob; // Convertit Fahrenheit en Celsius.
cout << p->recuperer_init() << " o Fahrenheit equivaut a ";
p->calculer();
cout << p->recuperer_conversion() << " o Celsius." << endl;
}
```

10.

```
#include <iostream>
#include <cstddef> // pour la définition de NULL
using namespace std ;
// ************* classes exceptions *************
class exc_liste {} ;
class exc_affec : public exc_liste {} ;
class exc_copie : public exc_liste {} ;
// ************* classe mere *************
class ParentMere
{ public :
virtual void affiche () = 0 ; // fonction virtuelle pure
} ;
// ************* classe liste *************
struct element // structure d'un element de liste
{ element * suivant ; // pointeur sur l'element suivant
ParentMere * contenu ; // pointeur sur un objet quelconque
} ;
class liste
{ element * debut ; // pointeur sur premier élément
element * courant ; // pointeur sur élément courant
public :
liste () // constructeur
{ debut = NULL ; courant = debut ; }
~liste () ; // destructeur
void ajoute (ParentMere *) ; // ajoute un élément
void premier () // positionne sur premier élément
{ courant = debut ;
}
ParentMere * prochain () // fournit l'adresse de l'élément
// courant (0 si fin)
// et positionne sur prochain
// élément (rien si fin)
{ParentMere * adsuiv = NULL ;
if (courant != NULL) { adsuiv = courant -> contenu ;
courant = courant -> suivant ;
}
return adsuiv ;
}
void affiche_liste () ; // affiche tous les éléments
// de la liste
int fini () { return (courant == NULL) ; }
liste & operator = (liste & l) { throw exc_affec() ; }
liste (liste &) { throw exc_copie() ; }
} ;
liste::~liste ()
{ element * suiv ;
courant = debut ;
```

```
while (courant != NULL )
{ suiv = courant->suivant ; delete courant ; courant = suiv ; }
}
void liste::ajoute (ParentMere * chose)
{ element * adel = new element ;

adel->suivant = debut ;
adel->contenu = chose ;
debut = adel ;
}
void liste::affiche_liste ()
{ ParentMere * ptr ;
premier() ;
while ( ! fini() )
{ ptr = (ParentMere *) prochain() ;
ptr->affiche () ;
}
}
// ************* classe point *************
class point : public ParentMere
{ int x, y ;
public :
point (int abs=0, int ord=0) { x=abs ; y=ord ; }
void affiche ()
{ cout << "Point de coordonnees : " << x << " " << y << "\n" ; }
} ;
// ************* classe complexe *************
class complexe : public ParentMere
{ double reel, imag ;
public :
complexe (double r=0, double i=0) { reel=r ; imag=i ; }
void affiche ()
{ cout << "Complexe : " << reel << " + " << imag << "i\n" ; }
} ;
// ************* programme d'essai *************
main()
{ try
{ liste l1 ;
point a(2,3), b(5,9) ;
complexe x(4.5,2.7), y(2.35,4.86) ;
l1.ajoute (&a) ; l1.ajoute (&x) ; l1.affiche_liste () ;
cout << "--------\n" ;
l1.ajoute (&y) ; l1.ajoute (&b) ; l1.affiche_liste () ;
liste l2 ;
l2 = l1 ; // provoque une exception exc_affec ;
}
catch (exc_liste)
{ cout << "tentative de copie ou d'affectation de liste" ;
}
}
```

11.

```
#include <iostream>
using namespace std ;
/* declaration (et définition) de la classe pile_entier */
/* ici, toutes les fonctions membres sont "inline" */
const int Max = 20 ;
class pile_entier
{ int dim ; // nombre maximal d'entiers de la pile
int * adr ; // adresse emplacement des dim entiers
int nelem ; // nombre d'entiers actuellement empiles
public :
pile_entier (int n = Max) // constructeur(s)
{ adr = new int [dim=n] ;
nelem = 0 ;
}
~pile_entier () // destructeur
{ delete adr ;
}
void empile (int p)
{ if (nelem < dim) adr[nelem++] = p ; }

int depile ()
{ if (nelem > 0) return adr[--nelem] ;
else return 0 ; // faute de mieux !
}
int pleine ()
{ return (nelem == dim) ; }
int vide ()
{ return (nelem == 0 ) ; }
} ;
/* programme d'essai de la classe pile_entier */
main()
{
int i ;
/* exemples d'utilisation de piles automatiques */
pile_entier a(3), // une pile de 3 entiers
b ; // une pile de 20 entiers (par défaut)
cout << "a pleine ? " << a.pleine () << "\n" ;
cout << "a vide ? " << a.vide () << "\n" ;
a.empile (3) ; a.empile (9) ; a.empile (11) ;
cout << "Contenu de a : " ;
for (i=0 ; i<3 ; i++) cout << a.depile () << " " ;
cout << "\n" ;
for (i=0 ; i<30 ; i++) b.empile (10*i) ;
cout << "Contenu de b : " ;
for (i=0 ; i<30 ; i++) if ( ! b.vide() ) cout << b.depile () << "
" ;
cout << "\n" ;
/* exemple d'utilisation d'une pile dynamique */
```

```
pile_entier * adp = new pile_entier (5) ;
// pointeur sur une pile de 5 entiers
cout << "pile dynamique vide ? " << adp->vide () << "\n" ;
for (i=0 ; i<10 ; i++) adp->empile (10*i) ;
cout << "Contenu de la pile dynamique : " ;
for (i=0 ; i<10 ; i++) if ( ! adp->vide() ) cout << adp->depile ()
<< " " ;
}
```