Université Pierre et Marie Curie — **Master M2 "IFMA"**
Programmation en C++ — **R. Roux, F. Benaych-Georges & D. Simon**

Chapitre 5 : Modèles de classes et de fonctions (*templates*), introduction à la STL, utilisation de bibliothèques extérieures, nouveau standard C++11

Table des matières

1 Modèles **1**
1.1 Présentation . . . 1
1.2 Exercices . . . 2

2 Introduction à la Standard Template Library **5**
2.1 Présentation générale . . . 5
2.2 Exercices . . . 5

3 Installation et utilisation de bibliothèques extérieures **12**
3.1 Qu'est-ce qu'une bibliothèque ? . . . 12
3.2 Installation des bibliothèques dans l'ordinateur . . . 12
3.3 Compilation et édition de liens pour des programmes faisant appel à des bibliothèques extérieures 12
3.3.1 Rappels sur la compilation et l'édition de liens sans bibliothèques extérieures . . . 12
3.3.2 Compilation avec bibliothèque extérieure . . . 13
3.3.3 Edition de liens avec bibliothèque extérieure . . . 13
3.3.4 Exemple . . . 13
3.4 Deux exemples de classes extérieures : les classes `array` et `random` de `Blitz++` . . . 14
3.5 Deux exemples de classes extérieures : les classes `uBLAS` et `random` de `Boost` . . . 15

4 Les nouveautés de C++11 **16**
4.1 Constructeurs . . . 16
4.2 Déplacement des objets . . . 17
4.3 Énumération dans des conteneurs de la STL . . . 18
4.4 Inférence de type . . . 18
4.5 Divers . . . 19

A Code nécessaire pour l'exercice 8 **20**

1 Modèles

1.1 Présentation

Un modèle définit une famille de fonctions ou de classes paramétrée par une liste d'identificateurs qui représentent des valeurs et des types. Les valeurs sont indiquées par leurs types respectifs, les types par le mot réservé `typename`. Le tout est préfixé par le mot réservé `template`.

Exemple :

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

template <typename T, int N>
void affichage_repete(T x) {for(int i=0; i<N; i++) {cout<<x<<endl;} };

template <typename S, int n>
class ma_classe { public : S membre;
        ma_classe(S membre) : membre(membre) {};
        void afficher() {cout<<n<<" "<<membre<<endl;}; };

int main() {ma_classe<double, 3> objet(3.14);
        objet.afficher();
        double var=3.14;
        affichage_repete<double, 5>(var); }
```

(nom du fichier : `Chapitre5/prog01.cpp`*)*

1.2 Exercices

Exercice 1 (Modèle de fonction sur un exemple simple)**.** *Que va afficher le programme suivant ? Que doit satisfaire le type* `T` *pour que* `tabulation_et_affichage<T>(x)`*, pour un élément* `x` *de type* `T`*, puisse être compilé ?*

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

template <typename T>
void tabulation_et_affichage(T x){cout<<"\t"<<x<<endl;};

int main() { string chaine="Bonjour";
        int n=7;
        double pi=3.14;
        tabulation_et_affichage<string>(chaine);
        tabulation_et_affichage<int>(n);
        tabulation_et_affichage(pi); }
```

(nom du fichier : `Chapitre5/prog02.cpp`*)*

Exercice 2 (Modèles de classes sur un exemple simple)**.** *Que va afficher le programme suivant ?*

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

template<typename le_type = int, int p = 1> class classe_exemple {
  public : classe_exemple(le_type x) : objet(x) {}
      void afficher() {cout<<"p vaut "<<p<<" et l\'objet est "<<objet<<endl;}
  private : le_type objet; };

int main () {
        classe_exemple<double, 5> x(1.5);
        x.afficher();

        classe_exemple<string, 7> y("...cette chaine !");
        y.afficher();

        classe_exemple<int, 10> z(4);
        z.afficher(); }
```

(nom du fichier : `Chapitre5/prog03.cpp`*)*

Exercice 3 (Modèles de classes)**.** *Sur le modèle de l'exemple précédent, créer une classe* `template` *qui permette de stocker des vecteurs à trois coordonnées de types éventuellement différents, de les additionner membre à membre,*

multiplier membre à membre, de les afficher et de calculer leur norme ℓ^p (rappel : pour $x \in \mathbb{R}^n$, la norme ℓ^p de x est $\|x\|_p = (\sum_i |x_i|^p)^{1/p}$).

Exercice 4 (Priorité des fonctions les plus spécialisées sur les fonctions les plus générales)**.** *Que va afficher le programme suivant ? (Indication : la réponse est contenue dans le titre de l'exercice...)*

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

template<typename T, typename R> T addition_generale (T x, R y)
{ cout<<"addition_TR"<<endl;  return x+y;};

template<typename T> T addition_generale (T x, T y)
{ cout<<"addition_T"<<endl;   return x+y;};

double addition_generale (double x, double y)
{ cout<<"addition_double"<<endl;  return x+y;};

int main () { int n=5,m=8;
        double x=1.3, y=2.5;
        addition_generale(x,y);
        addition_generale(n,m);
        addition_generale(x,n); };
```

(nom du fichier : `Chapitre5/prog04.cpp`*)*

Exercice 5 (Templates avec l'ordre lexicographique sur un tableau)**.** *Écrire une fonction rendant, pour un tableau (par exemple d'entiers ou de réels), le minimum du tableau, ayant le prototype suivant*

```
template<typename T> T min (T tab[], int n)
```

Que doit vérifier le type `T` *pour que la fonction puisse être instanciée avec le type* `T` *? La tester sur la classe suivante, que l'on aura écrite auparavant, avec* `<` *correspondant à l'ordre lexicographique. Écrire, de même, des fonctions*

```
template<typename T> int indice_max (T tab[], int n)
template<typename T> T lecture_element_max (T tab[], int n)
template<typename T> T &element_max (T tab[], int n)
```

rendant respectivement l'indice de la dernière apparition du maximum du tableau, l'élément maximum du tableau et une référence à l'élément maximum du tableau. Quel est la différence entre les fonctions `lecture_element_max` *et* `element_max` *? Tester ces fonctions.*

```
class notre_classe { public :
        notre_classe(double x=0, double y=0, int t=0) : a(x), b(y), c(t) {};
         friend bool operator<(const notre_classe &f, const notre_classe &g);
         friend  ostream& operator<<( ostream &, const notre_classe &);
 private:
         double a,b;
         int c; };
```

(nom du fichier : `Chapitre5/prog05.cpp`*)*

Exercice 6 (Template : modèle de fonction)**.** *Donner la sortie du programme suivant. Quel détail rend la ligne*

```
max(a,b)=5;
```

acceptable ? Peut-on décommenter l'avant-dernière ligne ?

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

template <typename T>T& max(T& a,T& b){return a>b?a:b;}

int main(){ int a=1,b=2;
    a=max(a,b);
    cout<<a<<endl;
    max(a,b)=5;
    cout<<b<<endl;
    string c="abc";
    string d="def";
    string e=max(c,d);
    cout<<e<<endl;
    //a=max(a,c);
    }
```

(nom du fichier : `Chapitre5/prog06.cpp`*)*

Exercice 7 (Utilisation des références et des templates)**.** *Comment modifier le programme suivant de façon à ce qu'il compile ? Que rendra-t-il alors ?*

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

template <typename T>T max(T& a,T& b){return a>b?a:b;}

int main() {
        double x=1, y=2;
        max(x,y)=0;
        cout<<y<<endl;
        return 0;}
```

(nom du fichier : `Chapitre5/prog07.cpp`*)*

Exercice 8 (Héritage, fonctions virtuelles, templates)**.** *Les classes définies ci-dessous sont une sophistication de celles d'un exercice fait au chapitre précédent. Voici quelques explications :*

- *Ici,* `Alea` *est la classe abstraite mère (elle contiendra tous les générateurs de variables aléatoires).*
- *Plus spécialisée (car elle hérite de la classe* `Alea`*),* `AleaDisc` *est une classe abstraite aussi (elle contiendra tous les générateurs de variables aléatoires* **discrètes** *à valeurs dans* $\mathbb{N}$*).*
  - *Dans* `AleaDisc`*, la méthode* `loi` *donne la loi (comme une fonction de* $\mathbb{N}$ *dans* $\mathbb{R}$*) de chaque objet.*
  - *Dans* `AleaDisc`*, la surcharge de l'opérateur* `()` *correspond au tirage du générateur de variables aléatoires. Sa définition est donnée ici, car elle ne dépend pas du générateur de variables aléatoires considéré, mais simplement de la méthode* `loi`*.*
  - *Les classes* `Uniform_disc` *(v.a. uniformes sur un intervalle),* `Poisson` *(v.a. de Poisson) et* `Bino` *(v.a. de loi binomiales) ne sont pas des classes abstraites et héritent de* `AleaDisc`*, donc de* `Alea`*. Il faut donc y déclarer et définir les méthodes* `esp`*,* `var` *et* `loi`*. Notons que la classe* `Bino` *aurait pu avoir simplement un membre* p *et un membre* N*, mais pour compliquer les choses, nous en avons fait une classe template, où* N *est un paramètre entier.*
- *La classe* `AleaCont` *est une classe abstraite aussi (elle contiendra tous les générateurs de variables aléatoires continues).*
  - •*] La classe* `AleaCont` *hérite de* `Alea`*, il faudra donc définir les méthodes* `esp`*,* `var` *et* `()` *dans toutes ses sous-classes non abstraites (notons que pour les v.a. sans espérance ou variance, on peut rendre un message d'erreur).*
  - *Dans* `AleaCont`*, la méthode* `dens` *donne la densité de la loi (comme une fonction de* $\mathbb{R}$ *dans* $\mathbb{R}$*) de chaque objet.*

Écrire ces classes, les tester. ***Le code est disponible à la fin de ce chapitre****.*

2 Introduction à la Standard Template Library

2.1 Présentation générale

La STL est un ensemble de modèles de classes destinées à représenter les structures les plus répandues (listes, vecteurs, ensembles,...) et qui possèdent des méthodes assez variées (tri, recherche d'un élément,...).

Voici, par exemple, comment on définit une liste d'entiers, un vecteur de doubles et un ensemble de points :

```
list<int> li;
vector<double> v;
set<point> s;
```

L'accès aux éléments contenus dans ces conteneurs est permis par ce que l'on appelle les *itérateurs*, qui généralisent les pointeurs (leur contenu est accessible via `*`, ils peuvent être incrémentés,...).

Exemple :

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

int main() {double t[] = {1,2,3,4,5,6};
        vector<double> v(t,t+6);
        vector<double>::iterator mon_iterateur = v.begin();
        cout<<*mon_iterateur<<endl;
        for(vector<double>::iterator it=++mon_iterateur;it!=v.end();++it)
                cout<<*it<<" ";
        cout<<endl;}
```

(nom du fichier : `Chapitre5/prog08.cpp`*)*

Une fois la notion d'*itérateur* digérée, l'utilisation de la STL ne repose plus que sur la connaissance des conteneurs et des commandes disponibles. Les exercices suivants permettront de découvrir ces conteneurs et ces commandes. En cas de doute, on consultera en particulier le site `http://www.cplusplus.com/reference/stl/`.

L'usage de ces structures accélère souvent l'exécution de programme car de nombreuses fonctionnalités sont implémentées de façon plus optimale que les méthodes naïves que l'on peut imaginer sans être spécialiste du sujet. Il convient de faire attention néanmoins : pour des exercices aussi basiques que l'énumération des nombres premiers du TP1, l'usage de la STL est "un rouleau-compresseur" pour écraser une mouche et ralentit considérablement le temps d'exécution.

2.2 Exercices

Exercice 9 (Utilisation des chaînes)**.** *Que va afficher le programme suivant ?*

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

int main() {
  string chaine1("ABC...");
  cout<<chaine1<<endl;
  string chaine2(" XYZ");
  chaine1+=chaine2;
  cout<<chaine1<<endl;
  return 0;}
```

(nom du fichier : `Chapitre5/prog09.cpp`*)*

Exercice 10 (STL : la classe vector)**.** *Que va afficher le programme suivant ?*

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

int main() {
        double t[]={1.,2.,3.};
        vector<double> v(t,t+3);
        vector<double>::iterator it=v.begin();
        for(;it!=v.end();++it) cout<<*it<<" ";
        cout<<endl;
        return 0;}
```

(nom du fichier : `Chapitre5/prog10.cpp`*)*

Exercice 11 (STL : iterateurs)**.** *Que va afficher le programme suivant ?*

```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

int main() {
        vector<pair<int, double> > v(5);
        for(int i=0;i<5;i++) {v[i]=make_pair(i,pow(2.,i));}
        vector<pair<int, double> >::iterator it=v.begin();
        for(;it!=v.end();++it) cout<<it->first<<" "<<it->second<<endl;
        return 0;}
```

(nom du fichier : `Chapitre5/prog11.cpp`*)*

Exercice 12 (STL : iterateurs, utilisation de typedef)**.** *Que va afficher le programme suivant ?*

```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

int main() { vector<pair<int, double> > v(5);
        for(int i=0;i<5;i++) {v[i]=make_pair(i,pow(2.,i));}
        typedef vector<pair<int, double> >::iterator type_iter;
        type_iter it=v.begin();
        for(;it!=v.end();++it) cout<<it->first<<" "<<it->second<<endl; }
```

(nom du fichier : `Chapitre5/prog12.cpp`*)*

Exercice 13 (Nombres complexes avec `pair`)**.** *Que va afficher le programme suivant ? Qu'est-ce qui distingue, dans la classe* `Complex`*, les méthodes* `Re()` *et* `Im()` *? Pourrait-on remplacer la commande du programme* `main()` *demandant* `z.Re()=5;` *par* `z.Im()=5;` *?*

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

class Complex { public : Complex(double x=0, double y=0) : p(make_pair(x,y)) {};
                ~Complex () {cout<<"utilisation du destructeur"<<endl;};
                double & Re() {return p.first;};
                double Im() {return p.second;};
        private : pair<double, double> p; };

int main(){ Complex z(3,8);
  cout<<z.Re()<<endl;
  z.Re()=5;
  cout<<z.Re()<<endl;
  {Complex x(4,8);
  cout<<x.Re()<<endl;}
  cout<<z.Im()<<endl; }
```

(nom du fichier : Chapitre5/prog13.cpp*)*

Exercice 14 (Utilisation des listes)**.** *Que vont afficher les programmes suivants ? Comment simplifier l'écriture du premier d'entre eux ?*

```
#include <list>
#include <iostream>

int main(){ std::list<int> my_list;
  my_list.push_back(5);
  my_list.push_back(4);
  my_list.push_back(3);
  my_list.push_back(2);
  my_list.push_back(1);
  my_list.pop_back();
  std::list<int>::const_iterator iter (my_list.begin());
  std::cout<<*iter<<std::endl;
  iter++;
  iter++;
  std::cout<<*iter<<std::endl;
  std::list<int>::const_iterator lit (my_list.begin()), lend(my_list.end());
  for(;lit!=lend;++lit) std::cout << *lit << ' ';
  std::cout << std::endl;  }
```

(nom du fichier : Chapitre5/prog14.cpp*)*

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

int main(){
 int chaine[]={1,2,3,4,5};
 list<int> my_list(chaine, chaine+5); //construction

 cout<<my_list.front()<<" "<<my_list.back()<<" "<<my_list.size()<<endl;

 my_list.pop_back();

 cout<<my_list.front()<<" "<<my_list.back()<<" "<<my_list.size()<<endl;

 list<int>::const_iterator lit (my_list.begin()), lend (my_list.end());
 for(;lit!=lend;lit++) cout << *lit << ' ';
 cout << endl;

 list<int>::iterator iter (my_list.end());
 iter--;
 cout<<*iter<<endl;
 my_list.insert(iter,45);
 cout<<*iter<<endl;

 iter++;
 for(int i=0;i<my_list.size();i++) {iter--; cout << *iter << ' '; }

 cout << endl << *my_list.begin()<<endl;
 my_list.insert(my_list.begin(),chaine, chaine+5);
 iter=my_list.begin();
 for(int i=0;i<my_list.size();i++) { cout << *iter << ' '; iter++;}
 cout << endl;
 return 0;}
```

(nom du fichier : `Chapitre5/prog15.cpp`*)*

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

bool superieur_a_3(int n) {return n>3;}

int main(){
 int chaine1[]={1,2,3,4,5,1,2,3,4,5};
 int chaine2[]={-1,2,7,8,9,10};
 list<int> list1(chaine1, chaine1+10), list2(chaine2, chaine2+6);
                          //(constructions)
 list2.pop_front();
 list2.push_front(-2);

 list<int>::iterator iter (list1.begin());
 for(int i=0;i<list1.size();i++) { cout << *iter << ' '; iter++;}
 cout << endl;

 iter=list2.begin();
 for(int i=0;i<list2.size();i++) { cout << *iter << ' '; iter++;}
 cout << endl;

 list1.remove(2);
 iter=list1.begin();
 for(int i=0;i<list1.size();i++) { cout << *iter << ' '; iter++;}
 cout << endl;

 list1.remove_if(superieur_a_3);
 iter=list1.begin();
 for(int i=0;i<list1.size();i++) { cout << *iter << ' '; iter++;}
 cout << endl;

 list1.sort();
 iter=list1.begin();
 for(int i=0;i<list1.size();i++) { cout << *iter << ' '; iter++;}
 cout << endl;

 list1.unique();
 iter=list1.begin();
 for(int i=0;i<list1.size();i++) { cout << *iter << ' '; iter++;}
 cout << endl;

 list1.merge(list2);

 iter=list2.begin();
 for(int i=0;i<list2.size();i++) { cout << *iter << ' '; iter++;}
 cout << endl;
 iter=list1.begin();
 for(int i=0;i<list1.size();i++) { cout << *iter << ' '; iter++;}
 cout << endl;

 list1=list2;
 iter=list1.begin();
 for(int i=0;i<list1.size();i++) { cout << *iter << ' '; iter++;}
 cout << endl;

 return 0;}
```

(nom du fichier : `Chapitre5/prog16.cpp`*)*

Exercice 15 (Utilisation des vecteurs)**.** *On reprend les programmes ci-dessus en remplaçant partout* `list` *par* `vector`*. Lesquels vont compiler ? Que va produire le programme ci-dessous ? Peut-on y remplacer* `vector` *par* `list` *?*

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

int main(){
  vector<double> vector1(10);
  double i=1;
  for(vector<double>::iterator it=vector1.begin(); it != vector1.end();
   ++it, ++i) *it=i*i;

  vector<double>::const_iterator iter=vector1.begin()+4;
  cout<<vector1[0]<<" "<<vector1[1]<<" "<<vector1[2]
  <<" "<<vector1[3]<<" "<<*iter<<endl; //impossible avec list

  vector<double> vector2(vector1.begin(),vector1.end());
  //recopie vector1 dans vector2

  vector<double> vector3(vector1);
  //recopie vector1 dans vector3

  vector<double> vector4(vector2.rbegin(),vector2.rend());
  //recopie vector2 dans vector4 dans l'ordre inverse

 iter=vector1.begin();
 for(int i=0;i<vector1.size();i++) { cout << *iter << ' '; iter++;}
 cout << endl;

 iter=vector4.begin();
 for(int i=0;i<vector4.size();i++) { cout << *iter << ' '; iter++;}
 cout << endl;

  return 0;}
```

(nom du fichier : `Chapitre5/prog17.cpp`*)*

Exercice 16 (Utilisation des fonctions)**.** *Que va produire le programme ci-dessous ?*

```cpp
#include <list>
#include <map>
#include <string>
#include <iostream>
using namespace std;

int main(){
  map<string, int>age;
  cout<<age.size()<<endl;
  age["Antoine"]=29;
  age["Anne"]=21;
  age["Pierre"]=59;
  age["Marie"]=61;
  cout<<age.size()<<" "<<age["Anne"]<<endl;

  list<pair<string, string> > liste_villes;
  liste_villes.push_back(make_pair("Cecile","Saint-Maur"));
  liste_villes.push_back(make_pair("Pauline","Creteil"));
  liste_villes.push_back(make_pair("Chloe","Paris"));
  liste_villes.push_back(make_pair("Laura","Montreuil"));
  map<string, string>ville(liste_villes.begin(),liste_villes.end());

  map<string, string>::iterator iter=ville.begin();
  while(iter!=ville.end()) {cout<<iter->first<<" : "
  <<iter->second<<endl; iter++;}
 cout<<endl;

  ville.insert(make_pair("Anne", "Rennes"));
  iter=ville.begin();
  while(iter!=ville.end()) {cout<<iter->first<<" : "
  <<iter->second<<endl; iter++;}
  cout<<endl;

  ville.erase("Cecile");
  ville.erase("Aurelie");
  iter=ville.begin();
  while(iter!=ville.end()) {cout<<iter->first<<" : "
  <<iter->second<<endl; iter++;}
  cout<<endl;

  ville.clear();
  cout<<ville.size()<<endl;
  return 0;}
```

(nom du fichier : `Chapitre5/prog18.cpp`*)*

Exercice 17 (Deux variantes de la classe Polymere)**.** *En utilisant la classe* `Particule`*, définie au TP3, écrire la classe de déclaration suivante. La tester. Faire ensuite de même en remplaçant le membre privé*

```
vector<Particule> nuage;
```

par le membre privé

```
vector<Particule*> nuage;
```

(on retirera alors la propriété `const` *aux particules arguments des constructeurs, car lorsque l'on entre dans un vecteur l'adresse d'un objet, le compilateur considère qu'il est susceptible d'être modifié).*

```
#ifndef Nuage_part_HPP
#define Nuage_part_HPP

class Nuage_part {
public :
        static int nombre_de_Nuage_parts;
        Nuage_part(const Particule &p);
        Nuage_part(const Nuage_part &pol, const Particule &p);
        Nuage_part(const Particule &p, const Nuage_part &pol);
        Nuage_part(const Nuage_part &f);
        ~Nuage_part();
        Nuage_part& operator=(const Nuage_part &d);
        void ajoute(const Particule &F);
        void enleve();
        int taille();
        friend ostream& operator<<(ostream &, const Nuage_part &);
        Nuage_part operator+( const Nuage_part &) ;
        Nuage_part operator+(const Particule &) ;
        void deplace(const double u, const double v);
        void accelere(const double u, const double v) ;
        void tourne_droite() ;
        void tourne_gauche() ;
private:
        vector<Particule> nuage;
};
#endif
```

(nom du fichier : `Chapitre5/Nuage_part_enonce.hpp`*)*

3 Installation et utilisation de bibliothèques extérieures

3.1 Qu'est-ce qu'une bibliothèque ?

Nous avons vu comment définir de nouveaux genres d'objets avec les classes, comment les spécialiser dans des types assez précis avec l'héritage, comment les paramétrer avec les modèles de fonctions (*templates*) et comment les manipuler dans des fonctions avec les différentes méthodes et fonctions amies des classes.

Ces possibilités assez vastes sont une des raisons du succès de C++. Cependant, pour de nombreux exemples d'objets relativement standards, ce travail a souvent déjà été fait par d'autres programmeurs, et via internet, on peut se procurer ces classes, regroupées dans ce qu'on appelle des *bibliothèques extérieures* (en anglais : *libraries*).

Pour les utiliser, il faut alors les télécharger, les installer (i.e. les compiler) et préciser à notre éditeur de liens le lieu où les trouver dans notre machine.

3.2 Installation des bibliothèques dans l'ordinateur

Pour la plupart des bibliothèques, la procédure, sur un ordinateur sous Linux ou un Mac (pour les PC, des émulateurs de Linux existent), est la suivante :

a) On télécharge un fichier compressé (`.gz`, `.bz2`...).

b) On décompresse le fichier.

c) Dans le répertoire ainsi créé, on exécute un fichier exécutable qui s'appelle `install`, `configure`, ou de suffixe `.sh` (par exemple, pour la bibliothèque `Boost`, le fichier est `bootstrap.sh` puis `bjam` (on exécute `bootstrap.sh` puis `bjam`), pour la bibliothèque `Blitz++`, le fichier est `configure` et pour la bibliothèque `Matrix Template Library`, le fichier est `configure`).

d) À l'exécution de ce fichier, un fichier nommé `Makefile` est créé : c'est celui-là qui commandera la compilation de la classe. On l'exécute avec `make` (c'est à dire que, dans le bon répertoire, on tape `make`).

3.3 Compilation et édition de liens pour des programmes faisant appel à des bibliothèques extérieures

3.3.1 Rappels sur la compilation et l'édition de liens sans bibliothèques extérieures

La compilation d'un programme auxiliaire (i.e. sans fonction `main`) `mon_prog_aux.cpp` se fait avec la commande

```
g++ -c mon_prog_aux.cpp
```

Un fichier objets `mon_prog_aux.o` est alors créé.

L'édition de liens (i.e. la création d'un exécutable) à partir d'un programme principal (i.e. avec une fonction `main`) `mon_prog_princ.cpp` et de fichiers objets auxiliaires

`mon_prog_aux_1.o`, `mon_prog_aux_2.o`, ..., `mon_prog_aux_n.o`

se fait avec la commande

```
g++ mon_prog_aux_1.o mon_prog_aux_2.o ... mon_prog_aux_n.o mon_prog_princ.cpp -o mon_executable.exe
```

Un fichier exécutable `mon_executable.exe` est alors créé.

3.3.2 Compilation avec bibliothèque extérieure

Supposons que l'on veuille compiler un programme `mon_prog.cpp` faisant appel à des classes

`ma_classe_ext_1`, ..., `ma_classe_ext_n`

de librairies extérieures, déclarées dans des fichiers

`ma_classe_ext_1.hpp`, ..., `ma_classe_ext_n.hpp`

Il faut, en préambule du programme, faire apparaître :

```
#include <ma_classe_ext_1.hpp>
.
.
#include <ma_classe_ext_n.hpp>
```

et, à la compilation, intercaler :

`-I/`*chemin menant à* `ma_classe_ext_1.hpp` ... `-I/`*chemin menant à* `ma_classe_ext_n.hpp`

entre `g++` et le nom du fichier à compiler.

On peut inclure la fin du chemin menant à `ma_classe_ext_k.hpp`dans le `#include` (et dans ce cas, on ne la met pas après `g++ -I/...`).

3.3.3 Edition de liens avec bibliothèque extérieure

Il faut faire apparaître

`-L/`*chemin menant à* `ma_classe_ext_1.o` ... `-L/`*chemin menant à* `ma_classe_ext_n.o`

entre `g++` et le nom du fichier à compiler.

3.3.4 Exemple

Considérons la bibliothèque extérieure `Boost`, configurée pour le système `MacOSX`. Un répertoire `boost_1_44_0` est créé à l'étape b) du paragraphe 3.2.

Supposons que le chemin menant à ce fichier, depuis la racine, soit

```
/Users/Droopy/Informatique/boost_1_44_0/
```

Alors pour compiler un programme `mon_prog_aux.cpp` utilisant la classe `random` de `Boost`, on mettra en préambule

```
#include <boost/random>
```

et on compilera avec

```
g++ -I/Users/Droopy/Informatique/boost_1_44_0/ -c mon_prog_aux.cpp
```

Si l'on veut ensuite éditer des liens via un programme `mon_prog_princ.cpp`, on tapera

```
g++ -L/Users/Droopy/Informatique/boost_1_44_0/stage/lib/ mon_prog_aux.o mon_prog_princ.cpp -o me.exe
```

Si la compilation et l'édition de liens, pour un programme `mon_prog.cpp`, se font en même temps, on tape

```
g++ -I/Users/Droopy/Informatique/boost_1_44_0/ -L/Users/Droopy/Informatique/boost_1_44_0/stage/lib/
mon_prog.cpp -o me.exe
```

On peut aussi, dans ce cas, utiliser le `Makefile` suivant

```
CPP = g++
CPPFLAGS = -I/Users/Droopy/Informatique/boost_1_44_0/ //compilation
LDFLAGS = -L/Users/Droopy/Informatique/boost_1_44_0/stage/lib/ //edition de liens

PROG = me.exe
SRCS = mon_prog.cpp
OBJS = $(SRCS:.cpp=.o)

$(PROG) : $(OBJS)
        $(CPP) $(LDFLAGS) $(OBJS) -o $(PROG)

mon_prog.o : mon_prog.cpp

mon_clean :
        rm -f $(PROG) $(OBJS)
```

NB : Pour un programme en `Blitz++`, cela donnerait :

```
CPP = g++
CPPFLAGS = -I/Users/Droopy/Informatique/blitz-0.9/ //compilation
LDFLAGS = -L/Users/Droopy/Informatique/blitz-0.9/lib/ //edition de liens

PROG = mon_ex
SRCS = random_demo.cpp
OBJS = $(SRCS:.cpp=.o)

$(PROG) : $(OBJS)
        $(CPP) $(LDFLAGS) $(OBJS) -o $(PROG)

random_demo.o : random_demo.cpp

monclean :
        rm -f $(PROG) $(OBJS)
```

3.4 Deux exemples de classes extérieures : les classes `array` et `random` de `Blitz++`

L'utilisation de ces classes est présentée dans les programmes suivants.

```
#include <iostream>
#include <blitz/array.h>
 using namespace std;
 using namespace blitz;

int main() {
Array<double, 2> A(3,4), B(4,7); // A et B sont des tableaux bidimensionnels
                        // de doubles, de tailles resp. 3x4 et 4x7
A=3; // toutes les valeurs de A sont 3
A(0,0)=0;
cout<<A<<" "<<B<<endl;}
```

(nom du fichier : `Chapitre5/prog19.cpp`*)*

```
#include <iostream>
#include <random/normal.h>
 using namespace std;
 using namespace ranlib;

int main() {Normal<double> ma_va(0,1); //ma_va est une v.a. N(0,1)
        cout<<ma_va.random()<<endl;} //affichage d'un appel a ma_va
```

(nom du fichier : `Chapitre5/prog20.cpp`*)*

3.5 Deux exemples de classes extérieures : les classes uBLAS et random de Boost

L'utilisation de ces classes est présentée dans les programmes suivants.

```
#include <boost/numeric/ublas/vector.hpp>
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/io.hpp>
#include <iostream>

using namespace boost::numeric::ublas;

int main () { vector<double> x(2);
      x(0) = 1; x(1) = 2;

      matrix<double> A(2,2);
      A(0,0) = 0; A(0,1) = 1;
      A(1,0) = 2; A(1,1) = 3;

      vector<double> y = prod(A, x);

      std::cout << A <<" "<< y << std::endl;}
```

(nom du fichier : Chapitre5/prog21.cpp*)*

```
#include <boost/random.hpp>
#include <ctime>
using namespace boost;

double SampleNormal (double mean=0, double sigma=1){
    // Create a Mersenne twister random number generator
    // that is seeded once with #seconds since 1970
    static mt19937 rng(static_cast<unsigned> (std::time(0)));

    // select Gaussian probability distribution
    normal_distribution<double> norm_dist(mean, sigma);

    // links random number generator to distribution, forming a function
    variate_generator<mt19937&, normal_distribution<double> >
normal_sampler(rng, norm_dist);

    // sample from the distribution
    return normal_sampler();}

int main() {for(int i=0;i<10;i++) std::cout<<SampleNormal()<<std::endl;}
```

(nom du fichier : Chapitre5/prog22.cpp*)*

```
#include <boost/numeric/ublas/vector.hpp>
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/io.hpp>
#include <iostream>
#include <boost/random.hpp>
#include <ctime>
using namespace boost;
using namespace boost::numeric::ublas;

matrix<double> GUE(int n){
    // Create a Mersenne twister random number generator
    // that is seeded once with #seconds since 1970
    static mt19937 rng(static_cast<unsigned> (std::time(0)));

    // select Gaussian probability distribution
    normal_distribution<double> norm_dist(0, 1./2);

    // links random number generator to distribution, forming a function
    variate_generator<mt19937&, normal_distribution<double> >
normal_sampler(rng, norm_dist);

    matrix<double> A(n,n);
    for(int i=0;i<n;i++){for(int j=0; j<n;j++) A(i,j)=normal_sampler();}
    return A+trans(A);}

int main () {std::cout << GUE(4) << std::endl;}
```

(nom du fichier : `Chapitre5/prog23.cpp`*)*

4 Les nouveautés de C++11

Une nouvelle version de C++ a été publiée récemment et contient de nombreuses améliorations et optimisations du code existant. Vous pouvez consulter les pages

— `http://en.wikipedia.org/wiki/C%2B%2B11` et `http://fr.wikipedia.org/wiki/C%2B%2B11`

— `http://fr.openclassrooms.com/informatique/cours/introduction-a-c-2011-c-0x`

et constater que de nombreux rêves sont maintenant devenus réalités, comme le montre les exemples ci-dessous. Apprendre cette nouvelle version de C++ **ne remplace pas** tout ce que vous connaissez déjà car la plupart des entreprises préfèrent utiliser des choses bien maîtrisées et dont la qualité du résultat est prouvé plutôt que de prendre trop de risques avec de trop nombreuses options nouvelles. Néanmoins, à terme, tous les compilateurs accepteront C++11 et tous les codes évolueront vers ce nouveau standard. Il est donc bon pour vous d'en connaître les bases afin de répondre à d'éventuelles questions en entretien et de montrer que vos connaissances sont à jour.

La compilation s'effectue de la même manière qu'avant (utilisation de `g++` ou `clang++`) avec l'ajout de l'option `-std=c++11` dans la ligne de compilation :

```
g++ -std=c++11 votrefichier.cpp
```

4.1 Constructeurs

Comme nous l'avons déjà vu précédemment, les constructeurs partagent souvent les mêmes morceaux de code au début (création des tableaux dynamiques etc) pour laisser ensuite la place à différentes manières de répartir l'information initiale. Nous avions vu également que les constructeurs ne pouvaient pas s'appeler entre eux. Maintenant cela est possible. Par exemple, le code suivant est désormais correct :

```
class A {
        protected:
                unsigned n;
                double * t;
        public:
                A(int n=0): n(n), t(new double[n]) {};
                A(const A & a): A(a.n) { for(unsigned i=0;i<n;i++) t[i]=a.t[i];};
/* Ancienne ecriture: A(const A & a): n(a.n), t(new double[n])
                        { for(unsigned i=0;i<n;i++) t[i]=a.t[i];};      */
};
```

(nom du fichier : `Chapitre5/prog24.cpp`*)*

Cela permet de mieux séparer la partie création de la partie initialisation des valeurs. L'inconvénient est qu'il faut faire attention à ne pas créer de boucles dans les appels (avec un premier constructeur qui appelle le deuxième qui appelle le premier par exemple).

4.2 Déplacement des objets

Une grosse innovation est la notion de déplacement des objets. Très souvent, il nous est arrivé d'écrire à la fin d'une fonction de prototype `A fonction(A a)` une ligne du type `return A(...);` afin de construire au dernier moment l'objet renvoyé. Lors de l'utilisation, on va souvent écrire `b=f(a);` lorsque l'on a surchargé l'affectation `=` pour la classe `A`. Le problème est alors que l'on *crée* un objet pour ensuite le *copier* dans `b` et ensuite l'*effacer*, une fois la fonction terminée. Il aurait mieux valu *déplacer* l'objet `b` (son adresse et tous ses attributs) pour qu'il pointe à présent vers ce nouvel objet de la mémoire.

Attention, on ne souhaite déplacer que des objets destinés à être effacer ou qui ne sont liés à aucun autre objet !

Pour cela il y a de nouveaux types `double &&`, `int &&`, etc, appelé *rvalue reference* dont la finalité est de pointer vers des valeurs qu'on souhaite récupérer à la place d'autres et la fonction `move` de prototype

```
template <class T> typename remove_reference<T>::type && move(T&& a)
```

qui donne à son résultat la valeur de son paramètre sans aucune copie (redirection vers une autre zone mémoire !). Bien évidemment, le prototype ne contient pas de `const` car l'objet `a` perd alors sa valeur car toute la zone mémoire est réadressée au résultat ! Il faut donc faire très attention.

Voyons plusieurs exemples. Voici deux version de la fonction d'échange :

```
template<typename T>
void echange(T & a, T & b) {
        T sauv(b);
        b=a;
        a=sauv;
}

template<typename T>
void echange11(T & a, T & b) {
        T && sauv=move(b);
        b=move(a);
        a=move(sauv);
}
```

(nom du fichier : `Chapitre5/prog25.cpp`*)*

Dans le premier cas, la variable `sauv` est créée par *copie* de `b` puis on a *recopié* par affectation `b` dans `a`, puis on a *recopié* `sauv` dans `a` puis finalement effacé `sauv`. Dans la nouvelle fonction, aucun constructeur de copie n'est appelé et aucune recopie n'est faite.

Cela donne un peu plus de travail car il faudra écrire de nouveaux constructeurs par déplacement et de nouveaux opérateurs d'affectation par *rvalue references* (cf. exercice ci-dessous) mais cela apporte un énorme gain de place mémoire et de temps d'exécution.

Exercice 18. *Qu'affiche le code suivant ? Faites également un dessin pour comprendre la gestion de la mémoire dans cette exemple.*

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

class A {
public:
        A(void) {cout << "Constr. par defaut" << endl;};
        A(const A &) {cout << "Constr. par copie" << endl;};
        A(A &&) {cout << "Constr. par deplacement" << endl;};
        ~A() {cout << "Destructeur"<< endl;};
        A & operator=(const A &) {cout << "Une affectation"<< endl;}
        A & operator=(A &&) {cout << "Rien"<< endl;}
};

void echange(A & a, A & b) {
        A sauv(b);
        b=a;
        a=sauv;
}

void echange11(A & a, A & b) {
        A && sauv=move(b);
        b=move(a);
        a=move(sauv);
}

int main(void){
        A a,b;
        cout << "A l'ancienne, sans move: " << endl << endl;
        echange(a,b);

        cout << endl << "Nouvelle methode, avec move: "<< endl << endl;
        echange11(a,b);

        cout << "Fin du programme." << endl;
        return 0;
}
```

(nom du fichier : `Chapitre5/prog26.cpp`*)*

4.3 Énumération dans des conteneurs de la STL

Pour parcourir un tableau statique ou dynamique, il fallait connaître son emplacement et sa taille pour utiliser cette dernière dans les paramètres d'itération de la boucle `for`. Cela reste bien évidemment toujours vrai pour les tableaux dynamiques. Néanmoins, pour tous les conteneurs plus évolués (`list`, `vector`, `array`, etc), la procédure était toujours la même : déclaration d'un itérateur, initialisation au début du conteneur, test de fin du conteneur. L'écriture d'une telle ligne reste néanmoins lourde et assez inintéressante. On pourra à présent directement parcourir la structure avec une ligne du type :

```
for(double & x: le_tableau_de_double) ...
```

et `x` vaudra successivement toutes les valeurs du tableau.

4.4 Inférence de type

Lorsque l'on manipule les itérateurs ou lorsque l'on utilise de manière assez lourde l'opérateur de résolution et les bibliothèques de type `boost`, écrire le type d'une variable peut se révéler fastidieux, bien que ce soit facile de décider ce type à partir des prototypes des fonctions appelées. Il y a à présent le mot-clef `auto` qui demande au compilateur de deviner le type de la variable à déclarer. Par exemple, dans le code

```
#include <iostream>

int main(void){
        int i=4;
        auto j=i;
        double x=5.14;
        auto y=x;
        auto u=new int[18];
}
```

(nom du fichier : `Chapitre5/prog27.cpp`*)*

les variables `j`, `y` et `u` auront pour type `int`, `double` et `int*`. Bien évidemment, l'usage de `auto` est interdit dans tout prototype de fonction ou de classe.

Cela est très pratique mais réserve malheureusement des surprises malheureuses. Tout d'abord, un abus de `auto` rend le code illisible et on ne sait plus qui est de quel type. De plus, dans le cadre de l'héritage de classes, on ne sait pas *a priori* quel type par défaut est choisi par `auto`. Son usage ne doit pas être généralisé par fainéantise mais doit rester restreint aux quelques cas où nul doute est possible.

Si on ne souhaite pas faire confiance à `auto`, alors il est également possible d'aller chercher le type d'une autre variable manuellement sans avoir à tout recopier à travers une syntaxe du type

```
decltype(i) j=4;
```

qui, dans l'exemple précédent, aurait mis un type `int` à la variable `j`.

4.5 Divers

Nous noterons également les améliorations suivantes :

— des fonctions nouvelles dans les conteneurs habituels de la STL et de nouveaux conteneurs `tuple` pour définir des produits cartésiens plus facilement.

— une génération des nombres aléatoires facilitée avec de nombreuses distributions standard prédéfinies.

— une généralisation du *multi-threading* pour améliorer le lancement parallèle d'exécution de fonctions sur des processeurs multi-cœurs.

Exercice 19. *Essayer de réécrire la plupart des codes que vous aviez écrits en utilisant ces nouveaux standards, en particulier l'utilisation de* `move` *pour optimiser au maximum l'utilisation de la mémoire.*

A Code nécessaire pour l'exercice 8

```
#ifndef ALEA_HPP_
#define ALEA_HPP_
#include <cstdlib>
#include <cmath>

inline double unif_rand() {
    return (random() + 0.5)/(RAND_MAX + 1.0);};

class Alea {
public:
    virtual double operator()() const = 0;
    virtual double esp() const = 0;
    virtual double var() const = 0;
    virtual ~Alea() {}
};

class AleaDisc : public Alea {
public:
    virtual double loi(int) const = 0;
    virtual double operator()() const;
    virtual ~AleaDisc() {}
};

double AleaDisc::operator()() const {
    int x = 0;
    double Fx = loi(0), y = unif_rand();
    while ( y > Fx ) {Fx += loi(++x);}
    return double(x);
};

class Uniform_disc : public AleaDisc {
        int a, b;
public :
        Uniform_disc(int, int) ;
        double loi(int ) const ;
        double esp() const ;
        double var() const ;
};

class Poisson : public AleaDisc {
    double lbda;
public:
    Poisson(double);
    double loi(int ) const   ;
    double esp() const ;
    double var() const ;
};

template<int N>
class Bino : public AleaDisc {
    double p;
    double Bnp[N+1];//stockera la loi
public:
    Bino(double );
    double loi(int ) const ;
    double esp() const ;
    double var() const ;
};
```

(nom du fichier : `Chapitre5/alea_enonce.hpp1`*)*

```
class AleaACont : public Alea {
public:
    virtual double dens(double) const = 0;
    virtual ~AleaACont() {} };

class Uniforme : public AleaACont {
public :
        Uniforme(double , double ) ;
        ~Uniforme(){};
        double dens(double ) ;
        double operator()() const ;
    double esp() const ;
    double var() const :
private :
        double  a,b; };

class Gauss : public AleaACont {
public :
        Gauss (double , double ) ;
        ~Gauss(){};
        double dens(double ) const;
        double operator()() const;
    double esp() const ;
    double var() const ;
private :
        double  m, sigmadeux; };

class Expo : public AleaACont {
public :
        Expo (double ) ;
        ~Expo(){};
        double dens(double ) ;
        double operator()() const ;
    double esp() const ;
    double var() const ;
private :
        double  lambda; };

class Cauchy : public AleaACont {
public:
    Cauchy() {}
    ~Cauchy(){}
    double dens(double ) const ;
    double operator()() const ;
    double esp() const ;
    double var() const ; };
#endif
```

(nom du fichier : `Chapitre5/alea_enonce.hpp2`*)*