Chapitre 5 : Tableaux

Exercice 5.1 Déclarations de tableaux classiques 1D (1)

Pour chacune des déclarations ci-dessous, indiquer si celle-ci est correcte ou non :

1)
```
int valeurs[10];
```

2)
```
const unsigned TAILLE = 10;
int valeurs[TAILLE];
```

3)
```
unsigned taille = 10;
int valeurs[taille];
```

4)
```
int valeurs[];
```

5)
```
int carres[5] = {0, 1, 4, 9, 16};
```

6)
```
int carres[] = {0, 1, 4, 9, 16};
```

7)
```
int carres[5] = {0, 1, 4, 9};
```

8)
```
int carres[5] = {0, 1, 4, 9, 16, 25};
```

9)
```
string noms[10];
```

Solution exercice 5.1

1) Correcte
2) Correcte
3) Incorrecte... tout du moins en C++ standard
   warning: ISO C++ forbids variable length array 'valeurs'
4) Incorrecte
   error: storage size of 'valeurs' isn't known
5) Correcte
6) Correcte.
   La taille du tableau est implicitement celle de l'agrégat
7) Correcte.
   Le tableau contiendra : 0 1 4 9 0 (les éléments non initialisés via l'agrégat sont mis à 0)
8) Incorrecte.
   error: too many initializers for 'int [5]'
9) Correct

Exercice 5.2 Déclarations de tableaux classiques 1D (2)

Ecrire les déclarations C++ correspondant aux énoncés suivants :
(si plusieurs formulations sont possibles, les proposer toutes)

1) Un tableau de 5 int
2) Un tableau de 5 int tel que le premier élément du tableau vaut 1, le second 2, etc.
3) Un tableau de 5 booléens tel que le premier élément du tableau vaut true et tous les autres éléments false

Solution exercice 5.2

1)
```
int t[5];
```

2)
```
int t[5] = {1, 2, 3, 4, 5};
int t[] = {1, 2, 3, 4, 5};
```

3)
```
bool t[5] = {true, false, false, false, false};
bool t[] = {true, false, false, false, false};
bool t[5] = {true};

NB toute valeur numérique != 0, resp == 0, peut faire l'affaire dans les
agrégats ci-dessus. Ainsi on pourrait par ex écrire :


bool t[5] = {1};
bool t[5] = {'A'};
bool t[5] = {3.14};
```

Exercice 5.3 Boucles et tableaux classiques 1D

En supposant que dans chacun des cas ci-dessous, le tableau *t* avant l'entrée dans la boucle est défini comme suit :

```
int t[] = {1, 2, 3, 4, 5, 4, 3, 2, 1, 0};
```

, déterminer ce que contient le tableau *t* au sortir de chacune des boucles :

1) `for (int i = 1; i < 10; ++i) { t[i] = t[i - 1]; }`

2) `for (int i = 9; i > 0; --i) { t[i] = t[i - 1]; }`

3) `for (int i = 0; i < 9; ++i) { t[i] = t[i + 1]; }`

4) `for (int i = 8; i >= 0; --i) { t[i] = t[i + 1]; }`

5) `for (int i = 1; i < 10; ++i) { t[i] = t[i] + t[i - 1]; }`

6) `for (int i = 1; i < 10; i = i + 2) { t[i] = 0; }`

7) `for (int i = 0; i < 5; ++i) { t[i + 5] = t[i]; }`

8) `for (int i = 1; i < 5; ++i) { t[i] = t[9 - i]; }`

Solution exercice 5.3

1) `1 1 1 1 1 1 1 1 1 1`
2) `1 1 2 3 4 5 4 3 2 1`
3) `2 3 4 5 4 3 2 1 0 0`
4) `0 0 0 0 0 0 0 0 0 0`
5) `1 3 6 10 15 19 22 24 25 25`
6) `1 0 3 0 5 0 3 0 1 0`
7) `1 2 3 4 5 1 2 3 4 5`
8) `1 1 2 3 4 4 3 2 1 0`

Exercice 5.4 Affichage de tableaux classiques 1D

Ecrire un programme C++ mettant à disposition une fonction permettant d'afficher n'importe quel tableau classique unidimensionnel d'entiers (type int) sous la forme :
`[valeur_1, valeur_2, …, valeur_n].`
NB L'affichage d'un tableau vide doit donner : `[]`

Tester votre fonction avec les tableaux suivants :

1) t0 = tableau vide
2) t1 = [1]
3) t2 = [1, 2]

Solution exercice 5.4

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

void afficher(const int tab[], size_t taille);

int main() {

   int t0[0]; // warning: ISO C++ forbids zero-size array 't0' [-pedantic]
   int t1[1] = {1};
   int t2[2] = {1,2};

   afficher(t0, 0);
   afficher(t1, 1);
   afficher(t2, 2);

   return EXIT_SUCCESS;
}

void afficher(const int tab[], size_t taille) {
   cout << "[";
   for (size_t i = 0; i < taille; ++i) {
      if (i > 0)
         cout << ", ";
      cout << tab[i];
   }
   cout << "]\n";
}
```

Exercice 5.5 Moyennes de notes

Ecrire un programme C++ qui :

1) demande à l'utilisateur de saisir une série de notes
2) calcule la moyenne de ces notes
3) affiche la moyenne de ces notes avec 2 chiffres après la virgule

Exemples d'exécution

```
Entrez la liste de vos notes (10 notes max), 0 pour quitter :
0

Moyenne non calculable car aucune note saisie !


Entrez la liste de vos notes (10 notes max), 0 pour quitter :
4
4.5
5.5
0

La moyenne des notes saisies = 4.67
```

Précisions

- La lecture des notes ainsi que le calcul de la moyenne sont à implémenter en tant que fonctions
- La vérification de la validité des notes saisies n'est pas à implémenter
- La lecture des notes doit se terminer automatiquement dès lors que l'utilisateur a saisi le nombre maximum de notes possibles

Solution exercice 5.5

```
#include <cassert>
#include <cstdlib>
#include <iomanip>
#include <iostream>
using namespace std;

/**
 @brief Lit une série de notes

 @param notes      le tableau dans lequel sont stockées les notes saisies
 @param nbNotesMax le nombre de notes maximal que peut stocker le tableau des notes
 @return           le nombre de notes saisies

 La validité des notes n'est pas vérifiée.
 La lecture se termine dès lors que l'utilisateur a saisi nbNotesMax notes.
*/
size_t lireNotes(double notes[], size_t nbNotesMax);

/**
 @brief Calcul de la moyenne des notes

 @param notes   le tableau des notes
 @param nbNotes le nombre de notes à moyenner
 @return        la moyenne des notes
*/
double moyenne(const double notes[], size_t nbNotes);

int main() {

   const size_t NB_NOTES_MAX = 10;
   double notes[NB_NOTES_MAX];
   double moyenneNotes;

   // Lecture des notes saisies par l'utilisateur
   size_t nbNotesSaisies = lireNotes(notes, NB_NOTES_MAX);

   cout << "nb de notes saisies = " << nbNotesSaisies << endl;

   cout << endl;
   if (nbNotesSaisies == 0) {
      cout << "Moyenne non calculable car aucune note saisie !" << endl;
   } else {
      // Calcul de la moyenne des notes
      moyenneNotes = moyenne(notes, nbNotesSaisies);
      // Affichage de la moyenne des notes
      cout << fixed << setprecision(2)
           << "La moyenne des notes saisies = " << moyenneNotes << endl;
   }

   return EXIT_SUCCESS;
}
```

```
size_t lireNotes(double notes[], size_t nbNotesMax) {
   assert(nbNotesMax != 0);

   cout <<  "Entrez la liste de vos notes"
        << " (" << nbNotesMax << " notes max), 0 pour quitter :"
        << endl;

   double noteSaisie;
   size_t nbNotesSaisies = 0;

   do {
      cin >> noteSaisie;
      if (noteSaisie != 0)
         notes[nbNotesSaisies++] = noteSaisie;
   } while (noteSaisie != 0 && nbNotesSaisies < nbNotesMax);

   return nbNotesSaisies;
}

double moyenne(const double notes[], size_t nbNotes) {
   assert(nbNotes != 0);
   double somme = 0;
   for (size_t i = 0; i < nbNotes; i++) {
      somme = somme + notes[i];
   }
   return somme / (double) nbNotes; // cast sinon warning
}
```

Autre variante possible pour *lireNotes* :

```
size_t lireNotes(double notes[], size_t nbNotesMax) {
   assert(nbNotesMax != 0);

   cout <<  "Entrez la liste de vos notes"
        << " (" << nbNotesMax << " notes max), 0 pour quitter :"
        << endl;

   for (size_t i = 0; i < nbNotesMax; ++i) {
      cin >> notes[i];
      if (notes[i] == 0)
         return i;
   }

   return nbNotesMax;
}
```

Exercice 5.6 Permutation du premier et du dernier élément

Ecrire une fonction C++ qui permute le premier et le dernier élément d'un tableau classique 1D d'entiers (type *int*).

Ecrire aussi un petit programme de test (*main*) permettant de vérifier le bon fonctionnement de la fonction ci-dessus.

Solution exercice 5.6

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

void permuterPremierEtDernierElement(int tab[], size_t taille);
void afficher(const int tab[], size_t taille);
void test(int tab[], size_t taille);

int main() {

   int t0[0]; // warning: ISO C++ forbids zero-size array 't0' [-pedantic]
   int t1[] = {1};
   int t2[] = {1, 2};
   int t3[] = {1, 2, 3};

   test(t0, 0);
   test(t1, 1);
   test(t2, 2);
   test(t3, 3);

   return EXIT_SUCCESS;
}

void permuterPremierEtDernierElement(int tab[], size_t taille) {
   if (taille > 1) {
      int tampon = tab[0];
      tab[0] = tab[taille - 1];
      tab[taille - 1] = tampon;
   }
}

void afficher(const int tab[], size_t taille) {
   cout << "[";
   for (size_t i = 0; i < taille; ++i) {
      if (i > 0)
         cout << ", ";
      cout << tab[i];
   }
   cout << "]\n";
}

void test(int tab[], size_t taille) {
   permuterPremierEtDernierElement(tab, taille);
   afficher(tab, taille);
}

// []
// [1]
// [2, 1]
// [3, 2, 1]
```

Exercice 5.7 Remplacement de valeurs par une valeur donnée

Sans faire usage d'une quelconque fonction prédéfinie, écrire une fonction C++ qui remplace toutes les valeurs paires d'un tableau classique 1D d'entiers (type *int*) par une valeur donnée.

Ecrire aussi un petit programme de test (*main*) permettant de vérifier le bon fonctionnement de la fonction ci-dessus.

Solution exercice 5.7

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

void remplacerValeursPaires(int tab[], size_t taille, int valeurRemplacement);
void afficher(const int tab[], size_t taille);
void test(int tab[], size_t taille, int valeurRemplacement);

int main() {

   int t0[0]; // warning: ISO C++ forbids zero-size array 't0' [-pedantic]
   int t1[] = {1};
   int t2[] = {1, 2};
   int t3[] = {1, 2, 3};

   test(t0, 0, 0);
   test(t1, 1, 0);
   test(t2, 2, 0);
   test(t3, 3, 0);

   return EXIT_SUCCESS;
}

void remplacerValeursPaires(int tab[], size_t taille, int valeurRemplacement) {
   for (size_t i = 0; i < taille; ++i) {
      if (tab[i] % 2 == 0) {
         tab[i] = valeurRemplacement;
      }
   }
}

void afficher(const int tab[], size_t taille) {
   cout << "[";
   for (size_t i = 0; i < taille; ++i) {
      if (i > 0)
         cout << ", ";
      cout << tab[i];
   }
   cout << "]\n";
}

void test(int tab[], size_t taille, int valeurRemplacement) {
   remplacerValeursPaires(tab, taille, valeurRemplacement);
   afficher(tab, taille);
}

// []
// [1]
// [1, 0]
// [1, 0, 3]
```

Exercice 5.8 Décalage à droite cyclique des éléments

Ecrire une fonction C++ qui décale d'une position vers la droite tous les éléments d'un tableau classique 1D d'entiers (type *int*), le dernier élément, lui, venant occuper la première position.

Ecrire aussi un petit programme de test (*main*) permettant de vérifier le bon fonctionnement de la fonction ci-dessus.

Solution exercice 5.8

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

void decalageDroiteCyclique(int tab[], size_t taille);
void afficher(const int tab[], size_t taille);
void test(int tab[], size_t taille);

int main() {

   int t0[0]; // warning: ISO C++ forbids zero-size array 't0' [-pedantic]
   int t1[] = {1};
   int t2[] = {1, 2};
   int t3[] = {1, 2, 3};

   test(t0, 0);
   test(t1, 1);
   test(t2, 2);
   test(t3, 3);

   return EXIT_SUCCESS;
}

void decalageDroiteCyclique(int tab[], size_t taille) {
   if (taille > 1) {
      int tmp = tab[taille - 1]; // Mémoriser le dernier élément
      for (size_t i = taille - 1; i > 0; --i) {
         tab[i] = tab[i - 1];
      }
      tab[0] = tmp;
   }
}

void afficher(const int tab[], size_t taille) {
   cout << "[";
   for (size_t i = 0; i < taille; ++i) {
      if (i > 0)
         cout << ", ";
      cout << tab[i];
   }
   cout << "]\n";
}


void test(int tab[], size_t taille) {
   decalageDroiteCyclique(tab, taille);
   afficher(tab, taille);
}

// []
// [1]
// [2, 1]
// [3, 1, 2]
```

Exercice 5.9 Suppression du ou des éléments centraux

Sans faire usage d'une quelconque fonction prédéfinie, écrire une fonction C++ qui supprime l'élément central d'un tableau classique 1D d'entiers (type *int*) si celui-ci est de taille impaire, respectivement les 2 éléments centraux si celui-ci est de taille paire.
L'ordre des éléments non supprimés doit être préservé.

Ecrire aussi un petit programme de test (*main*) permettant de vérifier le bon fonctionnement de la fonction ci-dessus.

Solution exercice 5.9

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

void supprimerElementsCentraux(int tab[], size_t& taille);
void afficher(const int tab[], size_t taille);
void test(int tab[], size_t taille);

int main() {

   int t0[0]; // warning: ISO C++ forbids zero-size array 't0' [-pedantic]
   int t1[] = {1};
   int t2[] = {1, 2};
   int t3[] = {1, 2, 3};
   int t4[] = {1, 2, 3, 4};
   int t5[] = {1, 2, 3, 4, 5};
   int t6[] = {1, 2, 3, 4, 5, 6};

   test(t0, 0);
   test(t1, 1);
   test(t2, 2);
   test(t3, 3);
   test(t4, 4);
   test(t5, 5);
   test(t6, 6);

   return EXIT_SUCCESS;
}

void supprimerElementsCentraux(int tab[], size_t& taille) {
   if (taille > 0) {
      const size_t NB_ELEM_CENTRAUX = taille % 2 ? 1 : 2;
      for (size_t i = taille / 2 + 1; i < taille; ++i)
         tab[i - NB_ELEM_CENTRAUX] = tab[i];
      taille -= NB_ELEM_CENTRAUX;
   }
}

void afficher(const int tab[], size_t taille) {
   cout << "[";
   for (size_t i = 0; i < taille; ++i) {
      if (i > 0)
         cout << ", ";
      cout << tab[i];
   }
   cout << "]\n";
}
```

```
void test(int tab[], size_t taille) {
   supprimerElementsCentraux(tab, taille);
   afficher(tab, taille);
}

// []
// []
// []
// [1, 3]
// [1, 4]
// [1, 2, 4, 5]
// [1, 2, 5, 6]
```

Exercice 5.10 Eléments strictement croissants ou pas ?

Sans faire usage d'une quelconque fonction prédéfinie, écrire une fonction C++ qui renvoie *true* si les éléments d'un tableau classique 1D d'entiers (type *int*) sont ordonnés de manière strictement croissante et *false* dans le cas contraire. On admettra ici qu'un tableau vide ou à 1 seul élément est ordonné de manière strictement croissante.

Ecrire aussi un petit programme de test (*main*) permettant de vérifier le bon fonctionnement de la fonction ci-dessus.

Solution exercice 5.10

Variante 1

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

bool estOrdonneStrictementCroissant(const int tab[], size_t taille);
void afficher(const int tab[], size_t taille);
void test(const int tab[], size_t taille);

int main() {

   int t0[0]; // warning: ISO C++ forbids zero-size array 't0' [-pedantic]
   int t1[] = {1};
   int t2[] = {1, 2};
   int t3[] = {1, 2, 3};
   int t4[] = {1, 2, 2};

   test(t0, 0);
   test(t1, 1);
   test(t2, 2);
   test(t3, 3);
   test(t4, 3);

   return EXIT_SUCCESS;
}

bool estOrdonneStrictementCroissant(const int tab[], size_t taille) {
   if (taille < 2)
      return true; // Convention
   for (size_t i = 0; i < taille - 1; ++i)
      if (tab[i] >= tab[i + 1])
         return false;
   return true;
}

void afficher(const int tab[], size_t taille) {
   cout << "[";
   for (size_t i = 0; i < taille; ++i) {
      if (i > 0)
         cout << ", ";
      cout << tab[i];
   }
   cout << "]";
}

void test(const int tab[], size_t taille) {
   afficher(tab, taille);
   cout << (estOrdonneStrictementCroissant(tab, taille) ? " est " : " n'est pas ")
        << "ordonne de maniere strictement croissante" << endl;
}

// [] est ordonne de maniere strictement croissante
// [1] est ordonne de maniere strictement croissante
// [1, 2] est ordonne de maniere strictement croissante
// [1, 2, 3] est ordonne de maniere strictement croissante
// [1, 2, 2] n'est pas ordonne de maniere strictement croissante
```

Variante 2 (plus naturelle… mais utilisant le type string qui n'est vu qu'au chapitre 6)

NB Dans plusieurs des exercices qui suivent, l'approche *toString* est préférée à l'approche *afficher*

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

bool estOrdonneStrictementCroissant(const int tab[], size_t taille);
string toString(const int tab[], size_t taille);
void test(const int tab[], size_t taille);

int main() {

   int t0[0]; // warning: ISO C++ forbids zero-size array 't0' [-pedantic]
   int t1[] = {1};
   int t2[] = {1, 2};
   int t3[] = {1, 2, 3};
   int t4[] = {1, 2, 2};

   test(t0, 0);
   test(t1, 1);
   test(t2, 2);
   test(t3, 3);
   test(t4, 3);

   return EXIT_SUCCESS;
}

bool estOrdonneStrictementCroissant(const int tab[], size_t taille) {
   if (taille < 2)
      return true; // Convention
   for (size_t i = 0; i < taille - 1; ++i)
      if (tab[i] >= tab[i + 1])
         return false;
   return true;
}

string toString(const int tab[], size_t taille) {
   string output;
   output += "[";
   for (size_t i = 0; i < taille; ++i) {
      if (i > 0)
         output += ", ";
      output += to_string(tab[i]);
   }
   output += "]";
   return output;
}

void test(const int tab[], size_t taille) {
   cout << toString(tab, taille)
        << (estOrdonneStrictementCroissant(tab, taille) ? " est " : " n'est pas ")
        << "ordonne de maniere strictement croissante" << endl;
}

// [] est ordonne de maniere strictement croissante
// [1] est ordonne de maniere strictement croissante
// [1, 2] est ordonne de maniere strictement croissante
// [1, 2, 3] est ordonne de maniere strictement croissante
// [1, 2, 2] n'est pas ordonne de maniere strictement croissante
```

NB Si la fonction prédéfinie to_string n'est pas disponible, il est possible d'écrire cette fonction soi-même, comme suit (nécessite #include <sstream>) :

```
string int_to_string(int n) {
   stringstream ss;
   ss << n;
   return ss.str();
}
```

Exercice 5.11 Somme alternée

Sans faire usage d'une quelconque fonction prédéfinie, écrire une fonction C++ qui calcule la *somme alternée* de tous les éléments d'un tableau classique 1D d'entiers (type *int*).

Exemple :
Si le tableau [1, 2, 3, 4] est passé en paramètre effectif à la fonction, celle-ci doit retourner le résultat de : 1 – 2 + 3 – 4, soit la valeur -2.

Ecrire aussi un petit programme de test (*main*) permettant de vérifier le bon fonctionnement de la fonction ci-dessus.

Remarques :
- On conviendra que la fonction renvoie 0 si le tableau passé en paramètre est vide.
- Il n'est pas demandé ici de tenir compte d'un éventuel débordement pouvant survenir lors du calcul de la somme alternée.

Solution exercice 5.11

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

int sommeAlternee(const int tab[], size_t taille);
string toString(const int tab[], size_t n);
void test(const int tab[], size_t taille);

int main() {

   int t0[0]; // warning: ISO C++ forbids zero-size array 't0' [-pedantic]
   int t1[] = {1};
   int t2[] = {1, 2, 3, 4};

   test(t0, 0);
   test(t1, 1);
   test(t2, 4);

   return EXIT_SUCCESS;
}

int sommeAlternee(const int tab[], size_t taille) {
   int sommeAlternee = 0;
   int coeff = 1;
   for (size_t i = 0; i < taille; ++i) {
      sommeAlternee += coeff * tab[i];
      coeff *= -1;
   }
   return sommeAlternee;
}

string toString(const int tab[], size_t n) {
   string output;
   output += "[";
   for (size_t i = 0; i < n; ++i) {
      if (i > 0)
         output += ", ";
      output += to_string(tab[i]);
   }
   output += "]";
   return output;
}

void test(const int tab[], size_t taille) {
   cout << "Somme alternee sur " << toString(tab, taille)
        << " = " << sommeAlternee(tab, taille) << endl;
}

// Somme alternee sur [] = 0
// Somme alternee sur [1] = 1
// Somme alternee sur [1, 2, 3, 4] = -2
```

Exercice 5.12 Suppression(s) d'une valeur donnée

Sans faire usage d'une quelconque fonction prédéfinie, écrire une fonction C++, sans valeur de retour, permettant de supprimer toutes les occurrences d'une valeur donnée dans un tableau classique 1D d'entiers (type *int*).
L'ordre des éléments non supprimés doit être préservé.

Ecrire aussi un petit programme de test (*main*) permettant de vérifier le bon fonctionnement de la fonction ci-dessus.

Solution exercice 5.12

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

void supprimer(int tab[], size_t& taille, int valeur);
void afficher(const int tab[], size_t taille);
void test(int tab[], size_t& taille, int valeur);

int main() {

   int t0[0]; // warning: ISO C++ forbids zero-size array 't0' [-pedantic]
   int t1[] = {1};
   int t2[] = {1, 2, 1, 1, 2, 1};

   size_t taille;

   taille = 0;
   test(t0, taille, 0);

   taille = 1;
   for (int i = 0; i <= 1; ++i) {
      test(t1, taille, i);
   }

   taille = 6;
   for (int i = 0; i <= 2; ++i) {
      test(t2, taille, i);
   }

   return EXIT_SUCCESS;
}

void supprimer(int tab[], size_t& taille, int valeur) {
   for (size_t i = 0; i < taille; ++i) {
      if (tab[i] == valeur) {
         for (size_t j = i + 1; j < taille; ++j) {
            tab[j - 1] = tab[j];
         }
         taille--;
         i--;
      }
   }
}
```

```
void afficher(const int tab[], size_t taille) {
   cout << "[";
   for (size_t i = 0; i < taille; ++i) {
      if (i > 0)
         cout << ", ";
      cout << tab[i];
   }
   cout << "]\n";
}

void test(int tab[], size_t& taille, int valeur) {
   supprimer(tab, taille, valeur);
   afficher(tab, taille);
}

// []
// [1]
// []
// [1, 2, 1, 1, 2, 1]
// [2, 2]
// []
```

A noter que la fonction *supprimer* proposée plus haut est peu efficace (décalage à gauche d'éléments à chaque fois que *valeur* est rencontrée).

Une façon plus efficace de faire consiste à écrire :

```
void supprimer(int tab[], size_t& taille, int valeur) {
   size_t j = 0;
   while (j < taille && tab[j] != valeur)
      ++j;
   for (size_t i = j; i < taille; ++i) {
      if (tab[i] != valeur) {
         tab[j++] = tab[i];
      }
   }
   taille = j;
}
```

Exercice 5.13 Suppression des doublons

Sans faire usage d'une quelconque fonction prédéfinie, écrire une fonction C++ permettant de supprimer tous les doublons figurant dans un tableau classique 1D d'entiers (type *int*).

Exemple :
Si le tableau [1, 2, 4, 2, 1, 1, 3] est passé en paramètre à la fonction, alors ce dernier doit se voir transformer en [1, 2, 4, 3].

Ecrire aussi un petit programme de test (*main*) permettant de vérifier le bon fonctionnement de la fonction ci-dessus.

Solution exercice 5.13

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

void supprimerDoublons(int tab[], size_t& taille);
void afficher(const int tab[], size_t taille);
void test(int tab[], size_t taille);

int main() {

   int t0[0]; // warning: ISO C++ forbids zero-size array 't0' [-pedantic]
   int t1[] = {1};
   int t2[] = {1, 2};
   int t3[] = {1, 2, 3};
   int t4[] = {1, 1, 1};
   int t5[] = {1, 2, 1};
   int t6[] = {1, 2, 1, 2};
   int t7[] = {1, 2, 3, 1, 2, 3};
   int t8[] = {1, 2, 3, 3, 2, 1};

   test(t0, 0);
   test(t1, 1);
   test(t2, 2);
   test(t3, 3);
   test(t4, 3);
   test(t5, 3);
   test(t6, 4);
   test(t7, 6);
   test(t8, 6);

   return EXIT_SUCCESS;
}

void supprimerDoublons(int tab[], size_t& taille) {
   // Si taille vaut 0 ou 1, inutile de faire quoi que ce soit
   if (taille > 1) {
      for (size_t i = 0; i < taille - 1; ++i) {
         for (size_t j = i + 1; j < taille; ++j) {
            if (tab[i] == tab[j]) {
               for (size_t k = j + 1; k < taille; ++k) {
                  tab[k - 1] = tab[k];
               }
               taille--;
               j--;
            }
         }
      }
   }
}

void afficher(const int tab[], size_t taille) {
   cout << "[";
   for (size_t i = 0; i < taille; ++i) {
      if (i > 0)
         cout << ", ";
      cout << tab[i];
   }
   cout << "]\n";
}
```

```
void test(int tab[], size_t taille) {
   supprimerDoublons(tab, taille);
   afficher(tab, taille);
}

// []
// [1]
// [1, 2]
// [1, 2, 3]
// [1]
// [1, 2]
// [1, 2]
// [1, 2, 3]
// [1, 2, 3]
```

A noter ici aussi (comme dans l'exercice 5.12) que la fonction *supprimerDoublons* proposée plus haut est peu efficace.

Une façon plus efficace de faire consiste à écrire :

```
void supprimer(int tab[], size_t& taille, int valeur, size_t pos = 0);
void supprimerDoublons(int tab[], size_t& taille);

void supprimer(int tab[], size_t& taille, int valeur, size_t pos) {
   if (pos < taille) {
      size_t j = pos;
      while (j < taille and tab[j] != valeur)
         ++j;
      for (size_t i = j; i < taille; ++i) {
         if (tab[i] != valeur) {
            tab[j++] = tab[i];
         }
      }
      taille = j;
   }
}

void supprimerDoublons(int tab[], size_t& taille) {
   if (taille > 1) {
      for (size_t i = 0; i < taille - 1; ++i) {
         supprimer(tab, taille, tab[i], i + 1);
      }
   }
}
```

Exercice 5.14 Egalité entre 2 tableaux (1)

Sans faire usage d'une quelconque fonction prédéfinie, écrire une fonction booléenne C++ qui détermine si 2 tableaux classiques 1D d'entiers (de type *int*) sont égaux[1] ou pas.

[1] On dira que 2 tableaux sont égaux s'ils sont tous deux vides ou s'ils possèdent les mêmes éléments (au sens ensembliste du terme). Par exemple, les 2 tableaux [3, 3, 1, 1, 2, 1] et [1, 2, 3] sont égaux car ils possèdent tous deux les mêmes éléments (1, 2 et 3).

Ecrire aussi un petit programme de test (*main*) permettant de vérifier le bon fonctionnement de la fonction ci-dessus.

NB N'hésitez pas à décomposer la fonction en "sous-fonctions" si nécessaire

Solution exercice 5.14

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

// Renvoie true si valeur se trouve dans le tableau tab.
// Renvoie false sinon.
bool appartientA(const int tab[], size_t taille, int valeur);

// Renvoie true si l'ensemble des valeurs du tableau tab1 est inclus dans
// l'ensemble des valeurs du tableau tab2.
// Renvoie false, sinon.
bool estInclus(const int tab1[], size_t taille1,
               const int tab2[], size_t taille2);

// Renvoie true si tab1 est égal à tab2.
// NB 2 tableaux sont égaux ssi ils sont tous deux vides ou s'ils possèdent les
// mêmes éléments (au sens ensembliste du terme)
bool sontEgaux(const int tab1[], size_t taille1,
               const int tab2[], size_t taille2);

string toString(const int tab[], size_t taille);

void test(const int tab1[], size_t taille1,
          const int tab2[], size_t taille2);

int main() {

   int t0[0]; // warning: ISO C++ forbids zero-size array 't0' [-pedantic]
   int t1[] = {1};
   int t2a[] = {1, 2};
   int t2b[] = {2, 1};
   int t3a[] = {1, 2, 3};
   int t3b[] = {3, 2, 1};
   int t4[] = {1, 2, 3, 4};
   int t5[] = {1, 2, 3, 2, 1};

   test(t0, 0, t0, 0);
   test(t1, 1, t1, 1);
   test(t2a, 2, t2b, 2);
   test(t3a, 3, t3b, 3);
   test(t3a, 3, t4, 4);
   test(t3a, 3, t5, 5);

   return EXIT_SUCCESS;
}
```

```
bool appartientA(const int tab[], size_t taille, int valeur) {
   for (size_t i = 0; i < taille; ++i) {
      if (tab[i] == valeur)
         return true;
   }
   return false;
}

bool estInclus(const int tab1[], size_t taille1,
               const int tab2[], size_t taille2) {
   for (size_t i = 0; i < taille1; ++i) {
      if ( !appartientA(tab2, taille2, tab1[i]) )
         return false;
   }
   return true;
}

bool sontEgaux(const int tab1[], size_t taille1,
               const int tab2[], size_t taille2) {
   return estInclus(tab1, taille1, tab2, taille2) &&
          estInclus(tab2, taille2, tab1, taille1);
}

string toString(const int tab[], size_t taille) {
   string output;
   output += "[";
   for (size_t i = 0; i < taille; ++i) {
      if (i > 0) {
         output += ", ";
      }
      output += to_string(tab[i]);
   }
   output += "]";
   return output;
}

void test(const int tab1[], size_t taille1, const int tab2[], size_t taille2) {
   cout << toString(tab1, taille1) << " et " << toString(tab2, taille2)
        << (sontEgaux(tab1, taille1, tab2, taille2) ? " sont " : " ne sont pas ")
        << "egaux" << endl;
}

// [] et [] sont egaux
// [1] et [1] sont egaux
// [1, 2] et [2, 1] sont egaux
// [1, 2, 3] et [3, 2, 1] sont egaux
// [1, 2, 3] et [1, 2, 3, 4] ne sont pas egaux
// [1, 2, 3] et [1, 2, 3, 2, 1] sont egaux
```

Exercice 5.15 Tous impairs ? (1)

Sans utiliser <*algorithm*>, écrire une fonction C++ qui prend en argument un vecteur (de type *vector*) de *int* et qui détermine si oui ou non tous les éléments du vecteur sont impairs.

NB Si le vecteur est vide, la fonction doit renvoyer *true*.

Ecrire également un petit programme de test (*main*) permettant de vérifier le bon fonctionnement de la fonction ci-dessus.

Solution exercice 5.15

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

bool sontTousImpairs(const vector<int>& v);
void test(const vector<int>& v);

int main() {
   test({});
   test({1, 2, 3});
   test({1, 3, 5});
   return EXIT_SUCCESS;
}

bool sontTousImpairs(const vector<int>& v) {
   for (int n : v) {
      if (n % 2 == 0) { // si présence élément pair
         return false;
      }
   }
   return true;
}

void test(const vector<int>& v) {
  cout << (sontTousImpairs(v) ? "Tous" : "Pas tous")
       << " les elements sont impairs";
   if (v.empty()) {
      cout << " (cas vecteur vide)";
   }
   cout << endl;
}

// Tous les elements sont impairs (cas vecteur vide)
// Pas tous les elements sont impairs
// Tous les elements sont impairs
```

Exercice 5.16 Concaténation de 2 vecteurs (1)

Sans utiliser <*algorithm*>, écrire une fonction C++ qui prend en paramètres 2 vecteurs (de type *vector*) de *int* et qui renvoie en valeur de retour le résultat de la concaténation de ces 2 vecteurs.

Ecrire également un petit programme de test (*main*) permettant de vérifier le bon fonctionnement de la fonction ci-dessus.

NB Appliquer le format suivant pour afficher un vecteur :
`[valeur_1, valeur_2,… valeur_n]` (`[]` pour un vecteur vide)

Exemple d'exécution

```
append([1, 3], [2, 4, 5]) = [1, 3, 2, 4, 5]
```

Solution exercice 5.16

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

vector<int> append(const vector<int>& v1, const vector<int>& v2);
void afficher(const vector<int>& v);
void test(const vector<int>& v1, const vector<int>& v2);

int main() {

   test({}, {});
   test({1}, {});
   test({1, 2}, {});
   test({}, {1});
   test({}, {1,2});
   test({1}, {2});
   test({1}, {2, 3});
   test({2,3}, {1});

   return EXIT_SUCCESS;
}

vector<int> append(const vector<int>& v1, const vector<int>& v2) {
   vector<int> v(v1);
   v.reserve(v1.size() + v2.size());
   for (int elem : v2)
      v.push_back(elem);
   return v;
}

void afficher(const vector<int>& v) {
   cout << "[";
   for (size_t i = 0; i < v.size(); ++i) {
      if (i > 0)
         cout << ", ";
      cout << v.at(i);
   }
   cout << "]";
}

void test(const vector<int>& v1, const vector<int>& v2) {
   cout << "append(";
   afficher(v1);
   cout << ", ";
   afficher(v2);
   cout << ") = ";
   afficher(append(v1, v2));
   cout << endl;
}

// append([], []) = []
// append([1], []) = [1]
// append([1, 2], []) = [1, 2]
// append([], [1]) = [1]
// append([], [1, 2]) = [1, 2]
// append([1], [2]) = [1, 2]
// append([1], [2, 3]) = [1, 2, 3]
// append([2, 3], [1]) = [2, 3, 1]
```

Autre variante (meilleure) :

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

vector<int> append(const vector<int>& v1, const vector<int>& v2);
string toString(const vector<int>& v);
void test(const vector<int>& v1, const vector<int>& v2);

int main() {

   test({}, {});
   test({1}, {});
   test({1, 2}, {});
   test({}, {1});
   test({}, {1,2});
   test({1}, {2});
   test({1}, {2, 3});
   test({2,3}, {1});

   return EXIT_SUCCESS;
}

vector<int> append(const vector<int>& v1, const vector<int>& v2) {
   vector<int> v;
   v.reserve(v1.size() + v2.size());
   v.insert(v.end(), v1.begin(), v1.end());
   v.insert(v.end(), v2.begin(), v2.end());
   return v;
}

string toString(const vector<int>& v) {
   string str;
   str += "[";
   for (size_t i = 0; i < v.size(); ++i) {
      if (i > 0)
         str += ", ";
      str += to_string(v.at(i));
   }
   str += "]";
   return str;
}

void test(const vector<int>& v1, const vector<int>& v2) {
   cout << "append(" << toString(v1) << ", " << toString(v2) << ") = "
        << toString(append(v1, v2)) << endl;
}
```

Exercice 5.17 Fusion alternée de 2 vecteurs

Sans utiliser <*algorithm*>, écrire une fonction C++ qui prend en paramètres 2 vecteurs (de type *vector*) de *int* et qui renvoie en valeur de retour le résultat de la fusion (*merge*) alternée des éléments des 2 vecteurs.

NB Si l'un des 2 vecteurs est plus long que l'autre, alterner les éléments aussi longtemps que possible, puis ajouter les éléments restants (du plus long des vecteurs).

Ecrire également un petit programme de test (*main*) permettant de vérifier le bon fonctionnement de la fonction ci-dessus.

Exemple d'exécution

```
merge([1, 3, 5, 7], [2, 4]) = [1, 2, 3, 4, 5, 7]
```

Solution exercice 5.17

```
#include <algorithm>
#include <cstdlib>
#include <iostream>
#include <string>
#include <vector>
using namespace std;

vector<int> merge(const vector<int>& v1, const vector<int>& v2);
string toString(const vector<int>& v);
void test(const vector<int>& v1, const vector<int>& v2);

int main() {

   test({}, {});
   test({1}, {});
   test({}, {1});
   test({1, 3}, {2, 4});
   test({1, 3, 5, 7}, {2, 4});
   test({1, 3}, {2, 4, 6, 8});

   return EXIT_SUCCESS;
}


vector<int> merge(const vector<int>& v1, const vector<int>& v2) {

   const size_t TAILLE_V1 = v1.size(),
                TAILLE_V2 = v2.size(),
                TAILLE_MIN = min(TAILLE_V1, TAILLE_V2),
                TAILLE_MAX = max(TAILLE_V1, TAILLE_V2);

   vector<int> v(TAILLE_V1 + TAILLE_V2);

   for (size_t i = 0; i < TAILLE_MIN; ++i) {
      v.at(2*i) = v1.at(i);
      v.at(2*i + 1) = v2.at(i);
   }

   if (TAILLE_V1 > TAILLE_V2) {
      for (size_t i = TAILLE_MIN; i < TAILLE_MAX; ++i)
         v.at(TAILLE_MIN + i) = v1.at(i);
   } else if (TAILLE_V2 > TAILLE_V1) {
      for (size_t i = TAILLE_MIN; i < TAILLE_MAX; ++i)
         v.at(TAILLE_MIN + i) = v2.at(i);
   }

   return v;
}

string toString(const vector<int>& v) {
   string str;
   str += "[";
   for (size_t i = 0; i < v.size(); ++i) {
      if (i > 0)
         str += ", ";
      str += to_string(v.at(i));
   }
   str += "]";
  return str;
}
```

```
void test(const vector<int>& v1, const vector<int>& v2) {
   cout << "merge(" << toString(v1) << ", " << toString(v2) << ") = "
        << toString(merge(v1, v2)) << endl;
}

// merge([], []) = []
// merge([1], []) = [1]
// merge([], [1]) = [1]
// merge([1, 3], [2, 4]) = [1, 2, 3, 4]
// merge([1, 3, 5, 7], [2, 4]) = [1, 2, 3, 4, 5, 7]
// merge([1, 3], [2, 4, 6, 8]) = [1, 2, 3, 4, 6, 8]
```

Autres solutions possibles :

```
vector<int> merge(const vector<int>& v1, const vector<int>& v2) {

   const size_t TAILLE_MIN = min(v1.size(), v2.size());

   vector<int> v(2 * TAILLE_MIN);

   for (size_t i = 0; i < TAILLE_MIN; ++i) {
      v.at(2*i) = v1.at(i);
      v.at(2*i + 1) = v2.at(i);
   }

   if (v1.size() > v2.size()) {
      v.insert(v.end(), v1.begin() + (int) TAILLE_MIN, v1.end());
   } else if (v2.size() > v1.size()) {
      v.insert(v.end(), v2.begin() + (int) TAILLE_MIN, v2.end());
   }

   return v;
}
```

(la plus "propre" de toutes)

```
vector<int> merge(const vector<int>& v1, const vector<int>& v2) {

   vector<int> v;
   v.reserve(v1.size() + v2.size());

   auto i1 = v1.begin();
   auto i2 = v2.begin();
   auto end = i1 + (long long) min(v1.size(), v2.size()); // cast sinon warning

   while (i1 != end) {
      v.push_back(*i1); ++i1;
      v.push_back(*i2); ++i2;
   }
   v.insert(v.end(), i1, v1.end());
   v.insert(v.end(), i2, v2.end());

   return v;
}
```

Exercice 5.18 Sommes des éléments diagonaux

Ecrire une fonction C++ qui retourne (par paramètres) la somme des éléments de la diagonale gauche[1] et la somme des éléments de la diagonale droite[1] d'une matrice carrée[2] à coefficients de type *int*.

[1] La diagonale *gauche* est la diagonale constituée des éléments allant de la valeur située sur la première ligne et la première colonne jusqu'à la valeur située sur la dernière ligne et la dernière colonne. La diagonale *droite* est la diagonale constituée des éléments allant de la valeur située sur la première ligne et la dernière colonne jusqu'à la valeur située sur la dernière ligne et la première colonne.

[2] La matrice carrée doit être implémentée sous la forme d'un *vector* bidimensionnel (2D).

Ecrire aussi un petit programme de test (*main*) permettant de vérifier le bon fonctionnement de la fonction ci-dessus.

Exemple

Si la matrice testée est :

$$M = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

, alors le programme de test devra afficher, à l'exécution :

```
La somme des elements de la diagonale gauche de [(1, 0, 1), (0, 1, 0), (1, 1, 0)] vaut 2
La somme des elements de la diagonale droite de [(1, 0, 1), (0, 1, 0), (1, 1, 0)] vaut 3
```

Solution exercice 5.18

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

using Matrice = vector<vector<int>>;

// On suppose que la matrice comprend au moins 1 élément et qu'elle est carrée
void sommesElementsDiagonaux(const Matrice& matrice,
                             int& sommeDiagGauche,
                             int& sommeDiagDroite);

string toString(const Matrice& matrice);

int main() {

   const Matrice MATRICE = { {1, 0, 1},
                             {0, 1, 0},
                             {1, 1, 0} };

   int sommeDiagGauche, sommeDiagDroite;

   sommesElementsDiagonaux(MATRICE, sommeDiagGauche, sommeDiagDroite);

   cout << "La somme des elements de la diagonale gauche de "
        << toString(MATRICE) << " vaut " << sommeDiagGauche << endl;
   cout << "La somme des elements de la diagonale droite de "
        << toString(MATRICE) << " vaut " << sommeDiagDroite << endl;

   return EXIT_SUCCESS;
}

void sommesElementsDiagonaux(const Matrice& matrice,
                             int& sommeDiagGauche,
                             int& sommeDiagDroite) {
   const size_t TAILLE = matrice.size();
   sommeDiagGauche = sommeDiagDroite = 0;
   for (size_t i = 0; i < TAILLE; ++i) {
      sommeDiagGauche += matrice[i][i];
      sommeDiagDroite += matrice[i][TAILLE - i - 1];
   }
}

string toString(const Matrice& matrice) {
   const size_t TAILLE = matrice.size();
   string output;
   output += "[";
   for (size_t i = 0; i < TAILLE; ++i) {
      if (i > 0) output += ", ";
      output += "(";
      for (size_t j = 0; j < TAILLE; ++j) {
         if (j > 0) output += ", ";
         output += to_string(matrice[i][j]);
      }
      output += ")";
   }
   output += "]";
   return output;
}
```

Exercice 5.19 Matrice de caractères

Ecrire un programme C++ qui :

1) crée et initialise un objet de la classe *vector* permettant de stocker la matrice (irrégulière) de caractères suivante :

```
abcdefghijklmnopqrstuvwxyz
abcdefghijklmnopqrstuvwxy
abcdefghijklmnopqrstuvwx
abcdefghijklmnopqrstuvw
abcdefghijklmnopqrstuv
abcdefghijklmnopqrstu
…
abc
ab
a
```

2) affiche à l'écran cette matrice

NB Les points 1) et 2) devront, chacun, être implémentés en tant que sous-programme.

Solution exercice 5.19

```
#include <cstdlib>
#include <iostream>
#include <numeric> // pour iota
#include <vector>
using namespace std;

using Matrice = vector<vector<char>>;

void initialiser(Matrice& m);
void afficher(const Matrice& m);

int main() {
   Matrice m;
   initialiser(m);
   afficher(m);
   return EXIT_SUCCESS;
}

void initialiser(Matrice& m) {
   const size_t NB_LIGNES = 26;
   m.clear(); // Nécess pour s'assurer que l'on part sur de bonnes bases
   m.resize(NB_LIGNES);
   for (size_t i = 0; i < NB_LIGNES; ++i) {
      m[i].reserve(NB_LIGNES - i);
      for (unsigned char c = 'a'; c <= 'z' - i; ++c) {
         m[i].push_back((char) c);
      }
   }
}

void afficher(const Matrice& m) {
   for (size_t i = 0; i < m.size(); ++i) {
      for (size_t j = 0; j < m[i].size(); ++j)
         cout << m[i][j];
      cout << endl;
   }
}
```

Autres variantes possibles :

```
void initialiser(Matrice& m) {
   const size_t NB_LIGNES = 26;
   m.resize(NB_LIGNES);
   for (size_t i = 0; i < NB_LIGNES; ++i) {
      const size_t NB_COLONNES = 26 - i;
      m[i].resize(NB_COLONNES);
      char c = 'a';
      for (size_t j = 0; j < NB_COLONNES; ++j) {
         m[i][j] = c++;
      }
   }
}
void initialiser(Matrice& m) {
   const string ALPHABET = "abcdefghijklmnopqrstuvwxyz";
   const size_t NB_LIGNES = 26;
   m.resize(NB_LIGNES);
   for (size_t i = 0; i < NB_LIGNES; ++i) {
      m[i].assign(ALPHABET.begin(),
                  ALPHABET.begin() + (long long) (NB_LIGNES - i));
   }
}

void initialiser(vector<vector<char>>& m) {
   size_t nbLignes = 26;
   m.resize(nbLignes);
   for (vector<char>& v : m) {
      v.resize(nbLignes--);
      iota(v.begin(), v.end(), 'a'); // iota se trouve dans <numeric>
   }
}
```

Remarque : On peut aussi résoudre ce problème comme suit :

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

using Matrice = vector<vector<char>>;

Matrice initialiser(const string& alphabet = "abcdefghijklmnopqrstuvwxyz");
void afficher(const Matrice& m);

int main() {
   Matrice m = initialiser();
   afficher(m);
   return EXIT_SUCCESS;
}

Matrice initialiser(const string& alphabet) {
   Matrice m(alphabet.size());
   for (size_t i = 0; i < alphabet.size(); ++i) {
      m.at(i).reserve(alphabet.size() - i);
      m.at(i).assign(alphabet.begin(),
                     alphabet.begin() + (long long) (alphabet.size() - i));
   }
   return m;
}

// Matrice initialiser(const string& alphabet) {
//    Matrice m;
//    m.reserve(alphabet.size());
//    for (size_t i = alphabet.size(); i > 0; --i)
//       m.emplace_back(alphabet.begin(), alphabet.begin() + (long long)i);
//    return m;
// }

void afficher(const Matrice& m) {
   for (auto& ligne : m) {
      for (auto c : ligne)
         cout << c;
      cout << endl;
   }
}
```

Exercice 5.20 Carré magique

Wikipédia *En mathématiques, un **carré magique d'ordre n** est composé de n^2 entiers strictement positifs, écrits sous la forme d'un tableau carré. Ces nombres sont disposés de sorte que leurs sommes sur chaque rangée, sur chaque colonne et sur chaque diagonale principale soient égales. On nomme alors **constante magique** (et parfois **densité**) la valeur de ces sommes.*
*Un **carré magique normal** est un cas particulier de carré magique, constitué de tous les nombres entiers de 1 à n^2, où n est l'ordre du carré.*

4	9	2
3	5	7
8	1	6

Un exemple de carré magique : carré magique normal d'ordre 3 et de constante magique 15.

L'algorithme de construction d'un carré magique normal d'ordre n impair est le suivant :
(l'algorithme de construction d'un carré magique normal d'ordre pair est plus compliqué)

```
noLigne = n – 1, noColonne = n / 2
Pour k = 1… n²
   Placer k en [noLigne][noColonne]
   Incrémenter noLigne et noColonne
   Si noLigne ou noColonne vaut n, remplacer sa valeur par 0
   Si l'élément en [noLigne][noColonne] a déjà été rempli
      Remettre noLigne et noColonne à leur valeur précédente
      Décrémenter noLigne
```

En tenant compte de ce qui précède, écrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir l'ordre (impair) du carré magique souhaité, détermine puis affiche à l'écran le contenu de ce carré magique.
NB Pensez à décomposer proprement le programme en sous-programmes.

Exemple d'exécution

```
Entrez l'ordre du carre magique souhaite (entier impair > 0) : a
Saisie incorrecte. Veuillez SVP recommencer.

Entrez l'ordre du carre magique souhaite (entier impair > 0) : 4
Saisie incorrecte. Veuillez SVP recommencer.

Entrez l'ordre du carre magique souhaite (entier impair > 0) : 5

11 18 25  2  9
10 12 19 21  3
 4  6 13 20 22
23  5  7 14 16
17 24  1  8 15
```

Solution exercice 5.20

```
#include <cmath>
#include <cstdlib>
#include <iomanip>
#include <iostream>
#include <limits>
#include <vector>
using namespace std;

using CarreMagique = vector<vector<unsigned>>;

unsigned lireOrdreCarreMagique(const string& msgInvite, const string& msgErreur);
CarreMagique carreMagique(unsigned ordreCarreMagique);
ostream& operator <<(ostream& os, const CarreMagique& carreMagique);

int main() {
   const unsigned ORDRE_CARRE_MAGIQUE =
      lireOrdreCarreMagique("Entrez un entier impair > 0 : ",
                            "Saisie incorrecte. Veuillez SVP recommencer.");
   cout << endl << carreMagique(ORDRE_CARRE_MAGIQUE) << endl;
   return EXIT_SUCCESS;
}

unsigned lireOrdreCarreMagique(const string& msgInvite, const string& msgErreur) {
   int ordreCarreMagique;
   bool saisieOk;
   do {
      cout << msgInvite;
      if (!(saisieOk = cin >> ordreCarreMagique
            && ordreCarreMagique > 0 && ordreCarreMagique % 2)) {
         cin.clear();
         if (!msgErreur.empty()) {
            cout << msgErreur << endl << endl;
         }
      }
      cin.ignore(numeric_limits<streamsize>::max(), '\n');
   } while (!saisieOk);
   return (unsigned) ordreCarreMagique;
}

CarreMagique carreMagique(unsigned ordreCarreMagique) {
   const unsigned NB_VALEURS = ordreCarreMagique * ordreCarreMagique;
   CarreMagique carreMagique(ordreCarreMagique,
                             vector<unsigned>(ordreCarreMagique));
   int ancienNoLigne,
       ancienNoColonne,
       noLigne = (int) ordreCarreMagique - 1,
       noColonne = (int) ordreCarreMagique / 2;
   for (unsigned n = 1; n <= NB_VALEURS; ++n) {
       carreMagique[(size_t) noLigne][(size_t) noColonne] = n;
       ancienNoLigne = noLigne++;
       ancienNoColonne = noColonne++;
       noLigne %= (int) ordreCarreMagique;
       noColonne %= (int) ordreCarreMagique;
       if (carreMagique[(size_t) noLigne][(size_t) noColonne] != 0) {
          noLigne = ancienNoLigne;
          noColonne = ancienNoColonne;
          noLigne--;
       }
   }
   return carreMagique;
}
```

```
ostream& operator <<(ostream& os, const CarreMagique& carreMagique) {
   const size_t ORDRE_CARRE_MAGIQUE = carreMagique.size();
   const int W = (int) log10(ORDRE_CARRE_MAGIQUE * ORDRE_CARRE_MAGIQUE) + 1;
   for (size_t noLigne = 0; noLigne < ORDRE_CARRE_MAGIQUE; ++noLigne) {
      for (size_t noColonne = 0; noColonne < ORDRE_CARRE_MAGIQUE; ++noColonne) {
         if (noColonne > 0) {
            os << ' ';
         }
         os << setw(W) << carreMagique[noLigne][noColonne];
      }
      os << endl;
   }
   return os;
}
```

Exercice 5.21 Conteneur array (tableau de taille fixe)

Ecrire un programme C++ qui :

1) Déclare et initialise (le plus simplement possible) un tableau de type *array* avec les valeurs 1, 2 et 3
2) Affiche (définir une fonction pour cela) le tableau
3) Affiche (sans utiliser ni l'opérateur [ ], ni la méthode `at`) la valeur du premier, respectivement du dernier, élément du tableau
4) Remet à 0 (sans utiliser ni boucle, ni agrégat) tous les éléments du tableau
5) Réaffiche le tableau
6) Se termine

Solution exercice 5.21

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

const size_t TAILLE = 3;

void afficher(const array<int, TAILLE>& arr);

int main() {

   array<int, TAILLE> monArray = {1, 2, 3}; // *
   afficher(monArray);

   cout << monArray.front() << " " << monArray.back() << endl;

   monArray.fill(0);
   afficher(monArray);

   return EXIT_SUCCESS;
}

void afficher(const array<int, TAILLE>& arr) {
   for (int elem : arr)
      cout << elem << " ";
   cout << endl;
}

// 1 2 3
// 1 3
// 0 0 0
```

(*) Egalement possible d'écrire : (dans les 2 cas, il y a appel au constructeur dit "initializer-list constructor")

```
array<int, TAILLE> monArray {1, 2, 3};
array<int, TAILLE> monArray({1, 2, 3})
```

Le code ci-dessus présente toutefois un défaut majeur : la fonction `afficher` dépend de la constante `TAILLE`… ce qui n'est pas souhaitable !

La solution consiste à recourir à la généricité et à la surcharge des opérateurs :

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

template <typename T, size_t N>
ostream& operator <<(ostream& os, const array<T, N>& arr);

int main() {

   const size_t TAILLE = 3;

   array<int, TAILLE> monArray = {1, 2, 3};
   cout << monArray;

   cout << monArray.front() << " " << monArray.back() << endl;

   monArray.fill(0);
   cout << monArray;

   return EXIT_SUCCESS;
}

template <typename T, size_t N>
ostream& operator <<(ostream& os, const array<T, N>& arr) {
   for (const T& elem : arr)
      os << elem << " ";
   os << endl;
   return os;
}
```

Exercice 5.22 Médailles olympiques

Soit le tableau suivant, donnant le nombre de médailles d'or, d'argent et de bronze obtenues par pays en patinage aux Jeux olympiques d'hiver de 2010 à Vancouver (Canada).

Pays	**Or**	**Argent**	**Bronze**
Canada	1	0	1
Chine	1	1	0
Allemagne	0	0	1
Corée	1	0	0
Japon	0	1	1
Russie	0	1	1
Etats-Unis	1	1	0

Ecrire un programme C++ mettant à disposition :

- une fonction retournant le nombre total de médailles obtenues par un pays donné
- une fonction retournant le nombre total de médailles d'un type donné obtenues par l'ensemble des pays (par ex le nombre total de médailles d'or obtenues par l'ensemble des pays)

… et bien sûr une fonction *main* permettant de tester les 2 fonctions précédentes.

NB Le tableau des médailles ci-dessus est à implémenter sous la forme d'un *array* bidimensionnel (2D)

Solution exercice 5.22

```
#include <array>
#include <cstdlib>
#include <iomanip>
#include <iostream>
#include <string>
using namespace std;

enum class Pays {CANADA, CHINE, ALLEMAGNE, COREE, JAPON, RUSSIE, ETATS_UNIS};
const size_t NB_PAYS = 7;
const string PAYS[] = {"Canada", "Chine", "Allemagne", "Coree",
                       "Japon", "Russie", "Etats-Unis"};

enum class TypeMedaille {OR, ARGENT, BRONZE};
const size_t NB_TYPES_MEDAILLE = 3;
const string TYPE_MEDAILLE[] = {"Or", "Argent", "Bronze"};

using Medailles_Obtenues = array<unsigned short, NB_TYPES_MEDAILLE>;
using Medailles = array<Medailles_Obtenues, NB_PAYS>;

// nombre total de médailles obtenues par un pays donné
size_t nbTotalMedailles(const Medailles& medailles,
                        const Pays& pays);

// nombre total de médailles d'un type donné obtenues par l'ensemble des pays
size_t nbTotalMedailles(const Medailles& medailles,
                        const TypeMedaille& typeMedaille);

int main() {

   const Medailles MEDAILLES = { Medailles_Obtenues{1, 0, 1},
                                 Medailles_Obtenues{1, 1, 0},
                                 Medailles_Obtenues{0, 0, 1},
                                 Medailles_Obtenues{1, 0, 0},
                                 Medailles_Obtenues{0, 1, 1},
                                 Medailles_Obtenues{0, 1, 1},
                                 Medailles_Obtenues{1, 1, 0} };

   // Nombre total de médailles obtenues par chacun des pays
   // NB Le setw(10) ci-dessous est fixé "en dur".
   //    Aurait pu (dû) se calculer en déterminant le nom du pays le plus long.
   for (Pays p = Pays::CANADA; p <= Pays::ETATS_UNIS;
        p = static_cast<Pays>(static_cast<int>(p) + 1)) {
      size_t nbMedailles = nbTotalMedailles(MEDAILLES, p);
      cout << setw(10) << left << PAYS[static_cast<int>(p)] << " : "
           << nbMedailles << " medaille" << (nbMedailles >= 2 ? "s" : "")
           << endl;
   }
   cout << endl;
```

```
   // Nombre total de médailles d'or, d'argent et de bronze obtenues
   // par l'ensemble des pays
   // NB Le setw(6) ci-dessous est fixé "en dur".
   //    Aurait pu (dû) se calculer en déterminant le nom du type de médaille
   //    le plus long.
   for (TypeMedaille tm = TypeMedaille::OR; tm <= TypeMedaille::BRONZE;
        tm = static_cast<TypeMedaille>(static_cast<int>(tm) + 1)) {
      size_t nbMedailles = nbTotalMedailles(MEDAILLES, tm);
      cout << setw(6) << left << TYPE_MEDAILLE[static_cast<int>(tm)] << " : "
           << nbMedailles << " medaille" << (nbMedailles >= 2 ? "s" : "")
           << endl;
   }

   return EXIT_SUCCESS;
}

size_t nbTotalMedailles(const Medailles& medailles,
                        const Pays& pays) {
   size_t nbTotalMedailles = 0;
   for (size_t j = 0; j < NB_TYPES_MEDAILLE; ++j) {
      nbTotalMedailles += medailles[static_cast<size_t>(pays)][j];
   }
   return nbTotalMedailles;
}

size_t nbTotalMedailles(const Medailles& medailles,
                        const TypeMedaille& typeMedaille) {
   size_t nbTotalMedailles = 0;
   for (size_t i = 0; i < NB_PAYS; ++i) {
      nbTotalMedailles += medailles[i][static_cast<size_t>(typeMedaille)];
   }
   return nbTotalMedailles;
}

// Canada     : 2 medailles
// Chine      : 2 medailles
// Allemagne  : 1 medaille
// Coree      : 1 medaille
// Japon      : 2 medailles
// Russie     : 2 medailles
// Etats-Unis : 2 medailles
//
// Or     : 4 medailles
// Argent : 4 medailles
// Bronze : 4 medailles
```

Exercice 5.23 Tous impairs ? (2)

Même énoncé que l'exercice 5.15 mais il est demandé ici de faire usage au maximum des fonctionnalités offertes par <*algorithm*>.

Solution exercice 5.23

Variante 1

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

bool estPair(int n);
bool estImpair(int n);
bool sontTousImpairs(const vector<int>& v);
void test(const vector<int>& v);

int main() {
   test({});
   test({1, 2, 3});
   test({1, 3, 5});
   return EXIT_SUCCESS;
}

bool estPair(int n) {
   return n % 2 == 0;
}

bool estImpair(int n) {
   return n % 2;
}

bool sontTousImpairs(const vector<int>& v) {
   return all_of(v.begin(), v.end(), estImpair);
   // Autre variante
//    return none_of(v.begin(), v.end(), estPair); // Faux d'écrire !estImpair
}


void test(const vector<int>& v) {
   cout << (sontTousImpairs(v) ? "Tous" : "Pas tous")
        << " les elements sont impairs";
   if (v.empty()) {
      cout << " (cas vecteur vide)";
   }
   cout << endl;
}
```

Variante 2 (avec expression lambda)

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

bool sontTousImpairs(const vector<int>& v);
void test(const vector<int>& v);

int main() {
   test({});
   test({1, 2, 3});
   test({1, 3, 5});
   return EXIT_SUCCESS;
}

bool sontTousImpairs(const vector<int>& v) {
   return all_of(v.begin(), v.end(), [](int n) {return n % 2;});
}

void test(const vector<int>& v) {
   cout << (sontTousImpairs(v) ? "Tous" : "Pas tous")
        << " les elements sont impairs";
   if (v.empty()) {
      cout << " (cas vecteur vide)";
   }
   cout << endl;
}
```

Rappel

```
template <class InputIterator, class UnaryPredicate>
  bool all_of(InputIterator first, InputIterator last, UnaryPredicate pred);
```

Returns `true` if *pred* returns `true` for all the elements in the range `[first,last)` or if the range is empty, and `false` otherwise.

Exercice 5.24 Concaténation de 2 vecteurs (2)

Même énoncé que l'exercice 5.16 mais il est demandé ici :

1) de faire usage au maximum des fonctionnalités offertes par <*algorithm*>,
2) d'implémenter la fonction d'affichage d'un vecteur en utilisant le concept d'itérateur.

Solution exercice 5.24

Variante avec copy + toString avec itérateur

```
#include <algorithm>
#include <cstdlib>
#include <iostream>
#include <string>
#include <vector>
using namespace std;

vector<int> append(const vector<int>& v1, const vector<int>& v2);
string toString(const vector<int>& v);
void test(const vector<int>& v1, const vector<int>& v2);

int main() {

   test({}, {});
   test({1}, {});
   test({1, 2}, {});
   test({}, {1});
   test({}, {1,2});
   test({1}, {2});
   test({1}, {2, 3});
   test({2,3}, {1});

   return EXIT_SUCCESS;
}

vector<int> append(const vector<int>& v1, const vector<int>& v2) {
   vector<int> v(v1.size() + v2.size());
   copy(v2.begin(), v2.end(), copy(v1.begin(), v1.end(), v.begin()));
   return v;
}

string toString(const vector<int>& v) {
   string str = "[";
   for (auto it = v.begin(); it != v.end(); ++it) {
      if (it != v.begin())
         str += ", ";
      str += to_string(*it);
   }
   str += "]";
   return str;
}

void test(const vector<int>& v1, const vector<int>& v2) {
   cout << "append(" << toString(v1) << ", " << toString(v2) << ") = "
        << toString(append(v1, v2)) << endl;
}
```

Variante avec copy_n + operator << avec itérateur

```
#include <algorithm>
#include <cstdlib>
#include <iostream>
#include <string>
#include <vector>
using namespace std;

ostream& operator <<(ostream& os, const vector<int>& v);
vector<int> append(const vector<int>& v1, const vector<int>& v2);
void test(const vector<int>& v1, const vector<int>& v2);

int main() {

   test({}, {});
   test({1}, {});
   test({1, 2}, {});
   test({}, {1});
   test({}, {1,2});
   test({1}, {2});
   test({1}, {2, 3});
   test({2,3}, {1});

   return EXIT_SUCCESS;
}

ostream& operator <<(ostream& os, const vector<int>& v) {
   os << '[';
   if (v.begin() != v.end()) {
      vector<int>::const_iterator it = v.begin();
      for (; it != prev(v.end()); ++it)
         os << *it << ", ";
      os << *it;
   }
   os << ']';
   return os;
}

vector<int> append(const vector<int>& v1, const vector<int>& v2) {
   vector<int> v(v1.size() + v2.size());
   copy_n(v2.begin(), v2.size(), copy_n(v1.begin(), v1.size(), v.begin()));
   return v;
}

void test(const vector<int>& v1, const vector<int>& v2) {
   cout << "append(" << v1 << ", " << v2 << ") = " << append(v1, v2) << endl;
}
```

NB La fonction *merge* de <algorithm> ne peut pas être utilisée ici car sa sémantique est la suivante : *"Combines the elements in the sorted ranges* `[first1,last1)` *and* `[first2,last2)`*, into a new range beginning at* result *with all its elements sorted".*

Ainsi, si notre fonction *append* était implémentée avec *merge*, on obtiendrait, par exemple :

```
append([1, 3], [2, 4, 5]) = [1, 2, 3, 4, 5]... au lieu de [1, 3, 2, 4, 5]
append([3, 1], [2, 4, 5]) = [2, 3, 1, 4, 5]... au lieu de [3, 1, 2, 4, 5]
```

Exercice 5.25 Egalité entre 2 tableaux (2)

Même énoncé que l'exercice 5.14 mais il est demandé ici 1) de considérer des vectors plutôt que des tableaux classiques, 2) de faire usage au maximum des fonctionnalités offertes par *<algorithm>*.

Solution exercice 5.25

Variante 1

```
#include <algorithm>
#include <cstdlib>
#include <iostream>
#include <string>
#include <vector>
using namespace std;

using Vecteur = vector<int>;

bool appartientA(const Vecteur& v, int val);
bool estInclus(const Vecteur& v1, const Vecteur& v2);
bool sontEgaux(const Vecteur& v1, const Vecteur& v2);

string toString(const Vecteur& v);
void test(const Vecteur& v1, const Vecteur& v2);

int main() {

   const Vecteur V0,
                 V1  = {1},
                 V2A = {1, 2},
                 V2B = {2, 1},
                 V3A = {1, 2, 3},
                 V3B = {3, 2, 1},
                 V4  = {1, 2, 3, 4},
                 V5  = {1, 2, 3, 2, 1};

   test(V0, V0);
   test(V1, V1);
   test(V2A, V2B);
   test(V3A, V3B);
   test(V3A, V4);
   test(V3A, V5);

   return EXIT_SUCCESS;
}

bool appartientA(const Vecteur& v, int val) {
   return find(v.begin(), v.end(), val) != v.end();
}

bool estInclus(const Vecteur& v1, const Vecteur& v2) {
   for (int i : v1)
      if (!appartientA(v2, i))
         return false;
   return true;
}
```

```
bool sontEgaux(const Vecteur& v1, const Vecteur& v2) {
   return estInclus(v1, v2) && estInclus(v2, v1);
}

string toString(const Vecteur& v) {
   string str = "[";
   for (auto i = v.begin(); i != v.end(); ++i) {
      if (i != v.begin())
         str += ", ";
      str += to_string(*i);
   }
   str += "]";
   return str;
}

void test(const Vecteur& v1, const Vecteur& v2) {
   cout << toString(v1) << " et " << toString(v2);
   if (sontEgaux(v1, v2)) {
      cout << " sont ";
   } else {
      cout << " ne sont pas ";
   }
   cout << "egaux" << endl;
}

// [] et [] sont egaux
// [1] et [1] sont egaux
// [1, 2] et [2, 1] sont egaux
// [1, 2, 3] et [3, 2, 1] sont egaux
// [1, 2, 3] et [1, 2, 3, 4] ne sont pas egaux
// [1, 2, 3] et [1, 2, 3, 2, 1] sont egaux
```

Variante 2

(A noter que cette variante n'a plus besoin des fonctions *appartientA* et *estInclus*)

```
Vecteur ensemble(const Vecteur& v) {
   Vecteur w(v);
   sort(w.begin(), w.end());
   w.erase(unique(w.begin(), w.end()), w.end());
   return w;
}

bool sontEgaux(const Vecteur& v1, const Vecteur& v2) {
   return ensemble(v1) == ensemble(v2);
}
```

Variante 3

(A noter que cette variante nécessite d'ajouter #include <set>)

```
bool sontEgaux(const Vecteur& v1, const Vecteur& v2) {
   return set<int>(v1.begin(), v1.end()) == set<int>(v2.begin(), v2.end());
}
```

Exercice 5.26 Algorithmes divers (1)

En exploitant au maximum les concepts d'itérateurs et d'algorithmes, écrire un programme C++ qui :

1. Déclare et initialise le tableau (classique) tab de *int* suivant : [3, 2, -5, 2, 4]
2. Déclare le vecteur v (de type *vector*) de *int* vide
3. Remplit v avec les valeurs contenues dans tab
4. Affiche v (en faisant : `cout << v`)
5. Affiche la plus petite valeur de v
6. Affiche la plus grande valeur de v
7. Affiche la somme des valeurs de v
8. Affiche le nombre d'occurrences de la valeur 2 dans v
9. Affiche combien v compte de valeurs impaires
10. Affiche v, trié dans l'ordre croissant
11. Affiche v, trié dans l'ordre décroissant
12. Affiche le vecteur construit en effectuant des sommes partielles sur les valeurs de v (cf exemple d'exécution ci-dessous)
13. Se termine

Votre programme doit reproduire, à l'exécution, le résultat suivant :

```
Le vecteur v initial :
[3, 2, -5, 2, 4]
La plus petite valeur de v : -5
La plus grande valeur de v : 4
La somme des elements de v : 6
Nombre d'occurrences de la valeur 2 dans v : 2
Nombre de valeurs impaires dans v : 2
Le vecteur v trie croissant :
[-5, 2, 2, 3, 4]
Le vecteur v trie decroissant :
[4, 3, 2, 2, -5]
Vecteur compose des sommes partielles de v :
[4, 7, 9, 11, 6]
```

Solution exercice 5.26

```
#include <algorithm>
#include <cstdlib>
#include <iostream>
#include <iterator> // pour back_inserter
#include <numeric>  // accumulate, partial_sum
#include <vector>
using namespace std;

ostream& operator<<(ostream& os, const vector<int>& v);
bool estImpair(int n);

int main() {

   const int TAB[] = {3, 2, -5, 2, 4};
   vector<int> v;

   const size_t TAILLE_TAB = sizeof(TAB) / sizeof(int);

   copy(TAB, TAB + TAILLE_TAB, back_inserter(v));
   cout << "Le vecteur v initial :\n";
   cout << v << endl;

   cout << "La plus petite valeur de v : "
        << *min_element(v.begin(), v.end()) << endl;

   cout << "La plus grande valeur de v : "
        << *max_element(v.begin(), v.end()) << endl;

   cout << "La somme des elements de v : "
        << accumulate(v.begin(), v.end(), 0) << endl;

   cout << "Nombre d'occurrences de la valeur 2 dans v : "
        << count(v.begin(), v.end(), 2) << endl;

   cout << "Nombre de valeurs impaires dans v : "
        << count_if(v.begin(), v.end(), estImpair) << endl;

   // Autre variante possible avec expression lambda
//    cout << "Nombre de valeurs impaires dans v : "
//         << count_if(v.begin(), v.end(), [] (int n) {return n % 2;}) << endl;

   sort(v.begin(), v.end());
   cout << "Le vecteur v trie croissant :\n";
   cout << v << endl;

   reverse(v.begin(), v.end()); // ou sort(v.rbegin(), v.rend());
   cout << "Le vecteur v trie decroissant :\n";
   cout << v << endl;

   vector<int> sp;
   partial_sum(v.begin(), v.end(), back_inserter(sp));
   cout << "Vecteur compose des sommes partielles de v :\n";
   cout << sp << endl;

   return EXIT_SUCCESS;
}
```

```
ostream& operator<<(ostream& os, const vector<int>& v) {
   os << '[';
   for (auto it = v.begin(); it != v.end(); ++it) {
      if (it != v.begin())
         os << ", ";
      os << *it;
   }
   return os << ']';
}

bool estImpair(int n) {
   return n % 2;
}
```

Exercice 5.27 Algorithmes divers (2)

En exploitant au maximum les concepts d'itérateurs et d'algorithmes, écrire un programme C++ qui :

1. Déclare et initialise le vecteur (de type *vector*) de *string* v suivant :
   v = [Pierre, Pierre, Pierre, Paul, Jacques, Jacques, Henri, Pierre, Paul, Jacques]
2. Affiche en quelles positions dans v figurent les prénoms "Paul" et "Henri"
3. Affiche en quelles positions dans v figure le motif formé des 3 prénoms "Pierre", "Paul" et "Jacques"
4. Affiche en quelles positions figurent des doublons (c'est-à-dire deux prénoms adjacents identiques)
5. Se termine

Votre programme doit reproduire, à l'exécution, le résultat suivant :

```
Dans le vecteur
[Pierre, Pierre, Pierre, Paul, Jacques, Jacques, Henri, Pierre, Paul, Jacques]
on trouve :


- Paul en position 3
- Henri en position 6
- Paul en position 8
- le motif [Pierre, Paul, Jacques] en position 2
- le motif [Pierre, Paul, Jacques] en position 7
- un doublon en position 0
- un doublon en position 1
- un doublon en position 4
```

Solution exercice 5.27

```
#include <algorithm>
#include <cstdlib>
#include <iostream>
#include <iterator>
#include <string>
#include <vector>
using namespace std;

ostream& operator <<(ostream& os, const vector<string>& v);

int main() {

   const vector<string> PRENOMS =
      {"Pierre", "Pierre", "Pierre", "Paul", "Jacques",
       "Jacques", "Henri", "Pierre", "Paul", "Jacques"};

   auto debut   = PRENOMS.begin(),
        fin     = PRENOMS.end(),
        courant = PRENOMS.begin();

   cout << "Dans le vecteur\n" << PRENOMS << "\non trouve :\n\n";

   // On recherche dans PRENOMS les positions des prénoms listés dans r1
   vector<string> r1 = {"Paul", "Henri"};
   while ((courant = find_first_of(courant, fin, r1.begin(), r1.end())) != fin) {
      cout << "- " << *courant
           << " en position " << distance(debut, courant) << endl;
      ++courant;
   }

   courant = PRENOMS.begin();

   // On recherche dans PRENOMS les positions du motif défini par r2
   vector<string> r2 = {"Pierre", "Paul", "Jacques"};
   while ((courant = search(courant, fin, r2.begin(), r2.end())) != fin) {
      cout << "- le motif " << r2
           << " en position " << distance(debut, courant) << endl;
      ++courant;
   }

   courant = PRENOMS.begin();

   // On recherche dans PRENOMS les positions des doublons
   // (càd de 2 prénoms adjacents identiques
   while ((courant = adjacent_find(courant, fin)) != fin) {
      cout << "- un doublon en position "
           << distance(debut, courant) << endl;
      ++courant;
   }

   return EXIT_SUCCESS;
}
```

```
ostream& operator <<(ostream& os, const vector<string>& v) {
   os << '[';
   if (v.begin() != v.end()) {
      vector<string>::const_iterator it = prev(v.end());
      for_each(v.begin(), it, [](string s) {cout << s << ", ";});
      os << *it;
   }
   os << ']';
   return os;
}

// Autre variante avec itérateur de sortie
// ostream& operator <<(ostream& os, const vector<string>& v) {
//    os << '[';
//    if (v.begin() != v.end()) {
//       ostream_iterator<string> sortie(os, ", ");
//       vector<string>::const_iterator it = prev(v.end());
//       copy(v.begin(), it, sortie);
//       os << *it;
//    }
//    os << ']';
//    return os;
// }
```