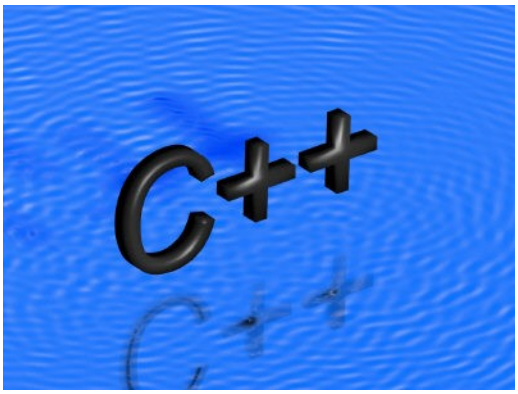
Hes·so
Haute Ecole Spécialisée
de Suisse occidentale

heig-vd
HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Département TIC

PRG1

Recueil d'exercices



René Rentsch

Table des matières

Chapitre 1 : Introduction 1
Exercice 1.1 Erreurs de syntaxe 1
Exercice 1.2 Un petit dessin 1
Exercice 1.3 Composition atmosphérique des planètes 2
Exercice 1.4 Votre année de naissance 2
Exercice 1.5 Train ou voiture ? 3
Exercice 1.6 Compte bancaire (1) 3
Exercice 1.7 Compte bancaire (2) 3
Chapitre 2 : Eléments de base 4
Exercice 2.1 Identificateurs C++ 4
Exercice 2.2 Déclarations de variables 5
Exercice 2.3 Types de base 5
Exercice 2.4 Taille et domaine de définition des types entiers 6
Exercice 2.5 Littéraux constants 6
Exercice 2.6 Evaluations d'expressions (1) 7
Exercice 2.7 Traduction d'expressions mathématiques 7
Exercice 2.8 Evaluations d'expressions (2) 8
Exercice 2.9 Conversions implicites 8
Exercice 2.10 Evaluations d'expressions (3) 9
Exercice 2.11 Volume d'une bouteille 9
Exercice 2.12 Conversion de mètres en miles, pieds et pouces 10
Exercice 2.13 Analyse d'un nombre entier (1) 10
Chapitre 3 : Structures de contrôle 11
Exercice 3.1 Frais de livraison 11
Exercice 3.2 Algorithme de résolution de $ax^2 + bx + c = 0$ 11
Exercice 3.3 If… else équivalents ou pas ? 11
Exercice 3.4 If…else dans tous leurs états 12
Exercice 3.5 Qui vaut 0 ? 13
Exercice 3.6 Evaluations d'expressions 13
Exercice 3.7 Pourquoi faire simple quand … ? 14
Exercice 3.8 Nombre de jours dans un mois donné (1) 14

Exercice 3.9 Minimum de 3 entiers 15
Exercice 3.10 Tri croissant de 3 entiers 15
Exercice 3.11 Multiple de 3 et/ou de 5 15
Exercice 3.12 Moyenne de notes et appréciation 16
Exercice 3.13 Instruction switch (1) 17
Exercice 3.14 Instruction switch (2) 18
Exercice 3.15 Nombre de jours dans un mois donné (2) 19
Exercice 3.16 Tracing manuel de variables 19
Exercice 3.17 Boucle while (1) 20
Exercice 3.18 Boucle while (2) 21
Exercice 3.19 Doublement d'un avoir bancaire 21
Exercice 3.20 Affichage de nombres par lots 22
Exercice 3.21 Boucles for (1) 22
Exercice 3.22 Boucles for (2) 23
Exercice 3.23 Boucles for (3) 23
Exercice 3.24 Balle qui rebondit 24
Exercice 3.25 Boucles for imbriquées 25
Exercice 3.26 Triangle d'étoiles 26
Exercice 3.27 Le code est-il simplifiable ? 26
Exercice 3.28 Série harmonique 27
Exercice 3.29 ppmc 27
Exercice 3.30 Simulations de boucles 28
Exercice 3.31 continue et break dans boucle do… while 28
Exercice 3.32 Approximation de π par la méthode Monte-Carlo 29
Exercice 3.33 Le problème des 3 portes 30
Chapitre 4 : Fonctions 31
Exercice 4.1 Passage par valeur 31
Exercice 4.2 Appels de fonctions 32
Exercice 4.3 Volume d'une pyramide à base rectangulaire 32
Exercice 4.4 Année bissextile 32
Exercice 4.5 Affichage de caractères compris entre 2 bornes 33
Exercice 4.6 Correction d'erreurs 33
Exercice 4.7 Mystère, mystère ! 34
Exercice 4.8 Permutation circulaire droite de 3 réels 34

Exercice 4.9 Retrait d'argent 35
Exercice 4.10 Fonction "Opérations arithmétiques" 35
Exercice 4.11 Passage par valeur vs par référence 36
Exercice 4.12 Portée des variables 37
Exercice 4.13 Fonction "compteur" 37
Exercice 4.14 Surcharge (1) 38
Exercice 4.15 Surcharge (2) 39
Exercice 4.16 Suite de Syracuse 40
Chapitre 5 : Tableaux 42
Exercice 5.1 Déclarations de tableaux classiques 1D (1) 42
Exercice 5.2 Déclarations de tableaux classiques 1D (2) 42
Exercice 5.3 Boucles et tableaux classiques 1D 43
Exercice 5.4 Affichage de tableaux classiques 1D 43
Exercice 5.5 Moyennes de notes 44
Exercice 5.6 Permutation du premier et du dernier élément 44
Exercice 5.7 Remplacement de valeurs par une valeur donnée 45
Exercice 5.8 Décalage à droite cyclique des éléments 45
Exercice 5.9 Suppression du ou des éléments centraux 45
Exercice 5.10 Eléments strictement croissants ou pas ? 45
Exercice 5.11 Somme alternée 46
Exercice 5.12 Suppression(s) d'une valeur donnée 46
Exercice 5.13 Suppression des doublons 46
Exercice 5.14 Egalité entre 2 tableaux (1) 47
Exercice 5.15 Tous impairs ? (1) 47
Exercice 5.16 Concaténation de 2 vecteurs (1) 47
Exercice 5.17 Fusion alternée de 2 vecteurs 48
Exercice 5.18 Sommes des éléments diagonaux 48
Exercice 5.19 Matrice de caractères 49
Exercice 5.20 Carré magique 50
Exercice 5.21 Conteneur array (tableau de taille fixe) 51
Exercice 5.22 Médailles olympiques 51
Exercice 5.23 Tous impairs ? (2) 52
Exercice 5.24 Concaténation de 2 vecteurs (2) 52
Exercice 5.25 Egalité entre 2 tableaux (2) 52

Exercice 5.26 Algorithmes divers (1) 53
Exercice 5.27 Algorithmes divers (2) 54
Chapitre 6 : Chaînes de caractères 55
Exercice 6.1 Analyse d'un caractère 55
Exercice 6.2 Constructeurs 56
Exercice 6.3 assign, '+', '+=' et append 57
Exercice 6.4 Affichage d'un petit dessin 58
Exercice 6.5 Vérifier qu'une chaîne de caractères est vide 58
Exercice 6.6 [ ], at, length, size, resize et substr 59
Exercice 6.7 Milieu d'une chaîne de caractères 60
Exercice 6.8 Inversion récursive d'une chaîne de caractères 60
Exercice 6.9 Numération romaine 61
Exercice 6.10 Analyse d'un nombre entier (2) 61
Exercice 6.11 Prénom, nom et acronyme (1) 62
Exercice 6.12 insert, replace et erase 63
Exercice 6.13 Compter le nombre d'occurrences 64
Exercice 6.14 Analyse d'une adresse 64
Exercice 6.15 Prénom, nom et acronyme (2) 65
Exercice 6.16 Saison correspondant à une date donnée 65
Exercice 6.17 Analyse d'un nombre entier (3) 66
Exercice 6.18 Conversions en base 8 et en base 16 66
Exercice 6.19 Saisie contrôlée d'un entier entre 2 bornes (1) 67
Exercice 6.20 Saisie contrôlée d'un entier entre 2 bornes (2) 68
Exercice 6.21 Saisie contrôlée d'un entier entre 2 bornes (3) 68
Chapitre 7 : Classes 69
Exercice 7.1 Point (1) 69
Exercice 7.2 Point (2) 69
Exercice 7.3 Pays 70
Exercice 7.4 Robot 70
Exercice 7.5 Point (3) 71
Exercice 7.6 Point (4) 71
Exercice 7.7 Surcharges d'opérateurs 72
Exercice 7.8 Membres statiques 73
Exercice 7.9 Voitures 74

Exercice 7.10 Messages 75
Exercice 7.11 Mailbox 76
Exercice 7.12 Personnes 77
Exercice 7.13 Constructeur de (re)copie et opérateur d'affectation 79
Chapitre 8 : Généricité 80
Exercice 8.1 Instanciations d'une fonction générique 80
Exercice 8.2 Occurrences d'une valeur dans un tableau 80
Exercice 8.3 Surcharge et spécialisation (1) 81
Exercice 8.4 Surcharge et spécialisation (2) 82
Exercice 8.5 Instanciations d'une classe générique 83
Exercice 8.6 Classe générique Array (1) 84
Exercice 8.7 Classe générique Array (2) 85
Exercice 8.8 Classe générique Array (3) 86
Exercice 8.9 Spécialisations 87
Exercice 8.10 Collection générique 88
Chapitre 9 : Exceptions 89
Exercice 9.1 try / catch (1) 89
Exercice 9.2 try / catch (2) 89
Exercice 9.3 try / catch (3) 90
Exercice 9.4 try / catch (4) 91
Exercice 9.5 Construction d'un objet membre (1) 92
Exercice 9.6 Construction d'un objet membre (2) 93
Exercice 9.7 Construction d'un objet membre (3) 93
Exercice 9.8 Construction d'un objet membre (4) 93
Exercice 9.9 Somme des n premiers entiers naturels (1) 94
Exercice 9.10 Somme des n premiers entiers naturels (2) 94
Exercice 9.11 Somme des n premiers entiers naturels (3) 94
Exercice 9.12 Insertion d'une valeur dans un tableau 94
Exercice 9.13 Exception avec info 95
Exercice 9.14 Terminaison de programme (1) 96
Exercice 9.15 Terminaison de programme (2) 97
Exercice 9.16 Terminaison de programme (3) ..... **Erreur ! Signet non défini.**

Chapitre 1 : Introduction

Exercice 1.1 Erreurs de syntaxe

Le programme ci-dessous contient plusieurs erreurs de syntaxe.

Proposez un correctif en regard de chaque ligne fautive.

	Correctif
`/* programme avec erreurs`	.....
`include iostream;`	.....
`use spacename std;`	.....
`int Main()`	.....
`   out < 'Hello' < endln;`	.....
`   Return;`	.....
`end;`	.....

Exercice 1.2 Un petit dessin

Ecrire un programme C++ permettant de reproduire à l'identique le dessin suivant :

```
  /////
 +------+
(| o o |)
 |  ^   |
 | '-' |
 +------+
```

Exercice 1.3 Composition atmosphérique des planètes

Notre système solaire est composé de 8 planètes : Mercure, Vénus, Terre, Mars, Jupiter, Saturne, Uranus et Neptune (dans l'ordre de leur distance au Soleil).

Les 4 premières planètes (Mercure, Vénus, Terre, Mars) sont dites "telluriques".

Les 4 dernières planètes (Jupiter, Saturne, Uranus et Neptune) sont dites "gazeuses".

Toutes les planètes, hormis Mercure, ont une atmosphère.

L'atmosphère de Vénus et de Mars est principalement constituée de CO_2 (dioxyde de carbone) et en moindre quantité de N_2 (diazote). Celle de la Terre est, elle, principalement constituée de N_2 et en moindre quantité de O_2 (dioxygène). Enfin, l'atmosphère des planètes gazeuses est, elle, principalement constituée de H_2 (dihydrogène) et en moindre quantité de He (hélium).

Ecrire un programme C++ qui affiche l'ensemble des informations données ci-dessus sous forme d'un tableau à 4 colonnes (type de la planète, nom de la planète, gaz principal et gaz secondaire).

Exercice 1.4 Votre année de naissance

Ecrire un programme C++ qui :

1. Demande à l'utilisateur d'entrer son prénom
2. Lit la réponse de l'utilisateur (on supposera celle-ci correcte) et la stocke dans une variable `prenom` de type *string*
3. Demande à l'utilisateur d'entrer son âge
4. Lit la réponse de l'utilisateur (on supposera celle-ci correcte) et la stocke dans une variable `age` de type entier *int*
5. Calcule l'année de naissance (à un an près) de l'utilisateur et l'enregistre dans une variable `annee_naissance` de type entier *int*
6. Affiche à l'écran le message suivant :
   `Bonjour <prenom>,`
   `Vous avez <age> ans et vous êtes ne(e) en <annee_naissance>.`

Indications

- Pour lire le prénom :
  - ajouter la ligne `#include <string>` avant le *main*
  - insérer les 2 lignes de code suivantes dans le *main*
    ```
    string prenom;
    getline(cin, prenom);
    ```
- Pour lire l'âge, utiliser l'instruction : `cin >> age;`

Exercice 1.5 Train ou voiture ?

Ecrire (en pseudo-code) l'algorithme permettant de traiter le problème suivant :

Vous souhaitez déterminer s'il est plus intéressant (du point de vue coût) de vous rendre de chez vous à votre lieu de travail en voiture ou en train. Les informations connues sont :

- *la distance en km séparant votre domicile de votre lieu de travail*
- *le coût du billet de train simple course*
- *la consommation (litres aux100 km) de la voiture*
- *le prix du litre d'essence*
- *les coûts d'amortissement (Frs par km) de la voiture*

Exercice 1.6 Compte bancaire (1)

Ecrire (en pseudo-code) l'algorithme permettant de traiter le problème suivant :

Initialement un compte bancaire possède un solde de 10'000.-. Le taux d'intérêt annuel de ce compte est de 6%. A la fin de chaque mois, les intérêts sont capitalisés et un retrait de 500.- est effectué. Au bout de combien de mois, le compte sera-t-il à découvert ?

Exercice 1.7 Compte bancaire (2)

Reprenons l'exercice 1.6 et supposons maintenant que le montant initial du compte, le taux d'intérêt annuel ainsi que le montant du retrait mensuel soient des données, non plus fixes, mais fournies par l'utilisateur.

Réécrire l'algorithme de l'exercice 1.6 en tenant compte de cette nouvelle donne.
NB On supposera que chacune des 3 valeurs fournies par l'utilisateur est ≥ 0.

Chapitre 2 : Eléments de base

Exercice 2.1 Identificateurs C++

Pour chacun des cas ci-dessous, indiquez s'il s'agit d'un identificateur C++ légal ou non. Justifiez votre réponse si celle-ci est "Non".

		Oui / Non	**Justification**
1)	007	……	……...
2)	james_bond_007	……	……...
3)	james_bond__007	……	……...
4)	james bond	……	……...
5)	sOs	……	……...
6)	SOS	……	……...
7)	_007	……	……...
8)	__007	……	……...
9)	_007_	……	……...
10)	bond-007	……	……...
11)	tom&jerry	……	……...
12)	int	……	……...
13)	INT	……	……...
14)	André	……	……...
15)	_	……	……...

Exercice 2.2 Déclarations de variables

a) Que vaut la variable *n* au terme de la séquence d'instructions suivante :

```
int n = 1;
n = 1 - 2 * n;
n = n + 1;
```

b) Expliquez pourquoi la séquence d'instructions suivante n'est pas correcte :

```
int n = 1;
n = n + 1;
int n = 1 - 2 * n;
```

c) Expliquez pourquoi la séquence d'instructions suivante n'est pas correcte :

```
int n = 1, p = 2;
n = (n + 1) * (n - P);
```

d) Expliquez pourquoi la séquence d'instructions suivante n'est pas correcte :

```
int n, m = 0;
n = 2 * n - 1;
m = n + 1;
```

Exercice 2.3 Types de base

1) Dressez la liste exhaustive de tous les types entiers supportés par C++
2) Idem pour les réels
3) Idem pour les caractères (à code non étendu)
4) Quel type permet en C++ de représenter des grandeurs booléennes ?
5) Quel type permet en C++ de représenter une absence de type ou un type neutre ?
6) Le type `int` est-il signé ou non signé par défaut ?
7) Le type `char` est-il signé ou non signé par défaut ?
8) Le domaine de définition des entiers est-il fixé par la norme ou dépend-il de l'environnement utilisé ?
9) Qu'ont de particulier les identificateurs des types de base en C++ ?

Exercice 2.4 Taille et domaine de définition des types entiers

Ecrire un programme C++ qui détermine / affiche à l'écran la taille en bits ainsi que l'intervalle des valeurs possibles des types *signed char*, *short*, *int*, *long* et *long long*.

Les résultats sont à présenter comme suit (ici pour le type *signed char*) :

```
signed char (8 bits) : -128 .. 127
```

Exercice 2.5 Littéraux constants

Indiquez si les littéraux constants suivants sont corrects ou non. Dans le cas où le littéral constant est correct, indiquez son type; dans le cas contraire, expliquez pourquoi il est faux.

1) 1.5 ……...

2) 1E3 ……...

3) 12u ……...

4) 12.0u ……...

5) 1L ……...

6) 1.0L ……...

7) .5 ……...

8) 5. ……...

9) 1000000000 ……...

10) 0x33 ……...

11) 0xefg ……...

12) 0xef ……...

13) 0xEF ……...

14) 0x0.2 ……...

15) 08 ……...

16) 07 ……...

Exercice 2.6 Evaluations d'expressions (1)

Que va afficher le programme C++ suivant ?

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

int main() {

   int i, j, k;

   i = 0; k = i++;
   cout << "A : i = " << i << " k = " << k << endl;

   i = 1; k = ++i;
   cout << "B : i = " << i << " k = " << k << endl;

   i = 2; j = 3;
   k = i++ * ++j;
   cout << "C : i = " << i << " j = " << j << " k = " << k << endl;

   i = 3; j = 4;
   k = i *= --j;
   cout << "D : i = " << i << " j = " << j << " k = " << k << endl;

   return EXIT_SUCCESS;
}
```

Exercice 2.7 Traduction d'expressions mathématiques

Ecrire l'équivalent C++ de chacune des expressions mathématiques ci-dessous :
(r, x et y sont supposés de type double; i, j et k de type int)

a) $\frac{4\pi r^3}{3}$

b) $\sqrt{|x-y|}$

c) $\sqrt{x^2+y^2}$

d) $\cos 45°$

e) e^{-x^2}

f) $\frac{i+j+k}{3}$

Exercice 2.8 Evaluations d'expressions (2)

Que va afficher le programme ci-dessous ? Expliquer les résultats obtenus.

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

int main() {

   cout << fixed << setprecision(0);

   cout << "1)" << 3 * 1000 * 1000 * 1000 << endl;
   cout << "2)" << 3.0 * 1000 * 1000 * 1000 << endl;
   cout << "3)" << 100000 * 100000 * 100000.0 << endl;
   cout << "4)" << 100000.0 * 100000 * 100000 << endl;
   cout << "5)" << 1E7 + 1.0 << endl;
   cout << "6)" << 1E7f + 1.f << endl;
   cout << "7)" << 1E8 + 1.0 << endl;
   cout << "8)" << 1E8f + 1.f << endl;

   return EXIT_SUCCESS;
}
```

Exercice 2.9 Conversions implicites

Soient les déclarations suivantes :

```
char c = 'A';
int n = 7;
int a = -2;
unsigned b = 1;
long p = 10;
float x = 1.25f;
double z = 5.5;
```

Pour chacune des expressions suivantes, indiquez :

- combien de conversions implicites sont mises en œuvre et lesquelles
- ce que vaut l'expression et quel est le type du résultat

1) n + c + p
2) 2 * x + c
3) (char) n + c
4) (float) z + n / 2
5) a + b

Exercice 2.10 Evaluations d'expressions (3)

Soient les déclarations suivantes :

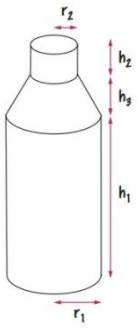
```
int i = 5, j = 11;
double x;
```

Que vaut la variable x dans chacun des cas ci-dessous ?

1) x = (double) j / i; ……...
2) x = double (j / i); ……...
3) x = j / i + .5; ……...
4) x = (double) j / i + .5; ……...
5) x = (int) (j + .5) / i; ……...

Exercice 2.11 Volume d'une bouteille

Comme illustré ci-dessous, la forme d'une bouteille peut s'approximer par deux cylindres, de rayons r_1 et r_2 et de hauteurs h_1 et h_2, joints par un cône tronqué de hauteur h_3.



Ecrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir (en centimètres) les cinq paramètres ci-dessus, calcule puis affiche (avec un chiffre après la virgule) la contenance en litres de la bouteille.

Exercice 2.12 Conversion de mètres en miles, pieds et pouces

Ecrire un programme C++ permettant de réaliser les trois conversions d'unités suivantes :

- mètres en miles
- mètres en pieds (*feet* en anglais)
- mètres en pouces (*inches* en anglais).

Le nombre de mètres est saisi par l'utilisateur sous la forme d'un entier (de type *unsigned int*) > 0. On suppose ladite saisie correcte.

Faire en sorte que l'affichage des résultats soit absolument identique à l'exemple d'exécution donné ci-dessous (alignement vertical des "=", résultats des conversions avec 2 chiffres après la virgule).

Exemple d'exécution

```
Entrez le nombre de metres a convertir (entier > 0) : 1000

1000 [m] = 0.62 [mile]
         = 3280.84 [ft]
         = 39370.08 [inch]
```

Exercice 2.13 Analyse d'un nombre entier (1)

Sans utiliser la classe *string* ni le concept de boucle (car pas encore vus en cours) mais en utilisant exclusivement les services offerts par la librairie *cmath*, écrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir un nombre entier (de type *unsigned int*) > 0, affiche à l'écran combien de chiffres contient ce nombre et ce que valent le premier chiffre et le dernier chiffre du nombre.

NB On supposera ici la saisie utilisateur correcte.

Exemple d'exécution

```
Entrez un nombre entier > 0 : 148

Nombre saisi       : 148
Nombre de chiffres : 3
Premier chiffre    : 1
Dernier chiffre    : 8
```

Chapitre 3 : Structures de contrôle

Exercice 3.1 Frais de livraison

Une société suisse tarife ses frais de livraison comme suit :

- 5 CHF si livraison en Suisse, à l'exception des cantons des Grisons et du Tessin où les frais de livraison se monte à 7 CHF
- 7 CHF si livraison au Liechtenstein
- 10 CHF partout ailleurs dans le monde

Représenter sous forme d'organigramme les diverses situations décrites ci-dessus.

Exercice 3.2 Algorithme de résolution de $ax^2 + bx + c = 0$

Représenter sous forme d'organigramme l'algorithme de résolution de l'équation :

$ax^2 + bx + c = 0; a, b, c \; et \; x \in R$

Exercice 3.3 If… else équivalents ou pas ?

Les deux extraits de code suivants sont-ils équivalents ? Justifier votre réponse.

1)
```
if (prixActuel > 100) {
   nouveauPrix = prixActuel - 20;
} else {
   nouveauPrix = prixActuel - 10;
}
```

2)
```
if (prixActuel < 100) {
   nouveauPrix = prixActuel - 10;
} else {
   nouveauPrix = prixActuel - 20;
}
```

Exercice 3.4 If…else dans tous leurs états

Quelle(s) remarque(s) vous inspirent les extraits de code suivants ?
NB Dans chacun des cas, proposez un correctif si vous estimez que l'extrait de code est mal écrit ou qu'il présente une erreur.
Dans tous les cas ci-dessous, on suppose bien sûr que prixActuel est tel que prixActuel – le rabais accordé est > 0.

1)
```
if (prixActuel >= 100) {
   nouveauPrix = prixActuel - 20;
   cout << "Nouveau prix = " << nouveauPrix << endl;
} else {
   nouveauPrix = prixActuel - 10;
   cout << "Nouveau prix = " << nouveauPrix << endl;
}
```

2)
```
if (prixActuel < 100) {
   nouveauPrix = prixActuel - 10;
} else if (prixActuel = 100) {
   nouveauPrix = prixActuel;
} else {
   nouveauPrix = prixActuel + 10;
}
```

3)
```
if (prixActuel < 100) {
   nouveauPrix = prixActuel - 10;
} else if (prixActuel >= 100 && prixActuel <= 150) {
   nouveauPrix = prixActuel - 15;
} else {
   nouveauPrix = prixActuel - 20;
}
```

4)
```
if (prixActuel < 100) {
   nouveauPrix = prixActuel - 10;
} else if (prixActuel >= 100) {
   nouveauPrix = prixActuel - 20;
}
```

5)
```
if (prixActuel >= 100) {
   droitAuRabais = true;
} else {
   droitAuRabais = false;
}
```

6)
```
if (prixActuel >= 100) {
   pourcentRabais = 5;
} else if (prixActuel > 500) {
   pourcentRabais = 10;
} else if (prixActuel > 1000) {
   pourcentRabais = 15;
} else {
   pourcentRabais = 0;
}
```

Exercice 3.5 Qui vaut 0 ?

On suppose disposer de deux entiers *x* et *y*.

Ecrire (de plusieurs manières différentes) la condition permettant de tester :

a) que nos deux entiers valent 0
b) qu'au moins l'un de nos deux entiers vaut 0
c) qu'un seul de nos deux entiers vaut 0

Exercice 3.6 Evaluations d'expressions

Que va afficher le programme C++ suivant ?

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

int main() {

   int i, j, k;

   i = j = k = 1;
   i += j += k;
   cout << "A : i = " << i << " j = " << j << " k = " << k << endl;

   i = 3; j = 2;
   k = i++ > j || j++ != 3;
   cout << "B : i = " << i << " j = " << j << " k = " << k << endl;

   i = 3; j = 2;
   k = i++ < j || j++ != 3;
   cout << "C : i = " << i << " j = " << j << " k = " << k << endl;

   i = 3; j = 2;
   k = ++i == 3 && ++j == 3;
   cout << "D : i = " << i << " j = " << j << " k = " << k << endl;

   i = 3; j = 2;
   k = ++i > 3 && ++j == 3;
   cout << "E : i = " << i << " j = " << j << " k = " << k << endl;

   return EXIT_SUCCESS;
}
```

Exercice 3.7 Pourquoi faire simple quand … ?

Trouver une formulation équivalente, **la plus courte et simple possible**, aux deux extraits de code ci-dessous :
(i, j et k sont supposés de type int et contenir une valeur parfaitement définie; b est supposé du type bool)

1)
```
if (j == 0) {
   b = true;
} else {
   if (i / j < k) {
      b = false;
   } else {
      b = true;
   }
}
```

2)
```
if (i < 1) {
   b = true;
} else {
   b = i > 2;
}
```

Exercice 3.8 Nombre de jours dans un mois donné (1)

Ecrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir un no de mois (1 pour janvier, 2 pour février, etc), affiche combien le mois choisi compte de jour.

IMPORTANT

- On suppose la saisie utilisateur correcte
- Le programme ne doit utiliser ni *if* ni *switch* mais doit exploiter les opérateurs logiques
- Si l'utilisateur entre la valeur 2, le programme doit répondre :
  "Ce mois comporte 28 ou 29 jours"

Exemple d'exécution

```
Entrez un no de mois (1-12) : 5
Ce mois comporte 31 jours.
```

Exercice 3.9 Minimum de 3 entiers

Ecrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir (sur une même ligne) 3 entiers (de type *int*) détermine / affiche le minimum de ces 3 entiers.

IMPORTANT
- On suppose la saisie utilisateur correcte
- Le problème doit être résolu sans utiliser d'autres librairies que *cstdlib* et *iostream*

Exercice 3.10 Tri croissant de 3 entiers

Ecrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir (sur une même ligne) 3 entiers (de type *int*) affiche la valeur de ces derniers dans l'ordre croissant.

IMPORTANT
- On suppose la saisie utilisateur correcte
- Le problème doit être résolu sans utiliser la fonction prédéfinie *swap*

Exercice 3.11 Multiple de 3 et/ou de 5

Ecrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir un entier $n \geq 0$, affiche, en fonction de la valeur de n, l'une des 4 réponses ci-dessous :
- <*n*> est un multiple de 3
- <*n*> est un multiple de 5
- <*n*> est un multiple de 3 et de 5
- <*n*> n'est ni un multiple de 3 ni un multiple de 5

IMPORTANT
- On suppose la saisie utilisateur correcte

Exercice 3.12 Moyenne de notes et appréciation

Ecrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir (sur une même ligne) 4 notes (de type *double*) données au dixième de point affiche :

1°) la moyenne (au dixième de point) de ces 4 notes
2°) une appréciation dépendant de cette moyenne selon le barème suivant :

- **Insuffisant** si moyenne < 4.0
- **Moyen** si moyenne comprise dans l'intervalle [4.0; 4.5]
- **Bien** si moyenne comprise dans l'intervalle (4.5; 5]
- **Très bien** si moyenne comprise dans l'intervalle (5.0; 5.5]
- **Excellent** si moyenne > 5.5

Exemple d'exécution

```
Entrez 4 notes : 5.7 4.3 5.2 4.4
Moyenne = 4.9 - Bien
```

IMPORTANT

- On suppose travailler avec des notes comprises (bornes incluses) entre 1 et 6
- On suppose la saisie utilisateur correcte
- **Rappel**
  Pour afficher une valeur de type *double* avec 1 chiffre après la virgule, il faut :
  - faire un `#include <iomanip>` (nécessaire pour pouvoir utiliser le manipulateur *setprecision*)
  - écrire : `cout << fixed << setprecision(1) << valeur_de_type_double`
- Le problème doit être résolu sans utiliser d'autres librairies que *cstdlib, iomanip* et *iostream*

Exercice 3.13 Instruction switch (1)

Indiquer si les affirmations suivantes sont justes ou fausses.

Proposer un correctif si vous jugez l'affirmation fausse.

1) Après le mot réservé **case** on peut donner une liste de valeurs séparées par des virgules :

   ……

2) Après le mot réservé **case** on peut donner un intervalle de valeurs :

   ……

3) La branche "**default**" peut venir n'importe où dans la liste des cas :

   ……

4) Les cas dans les différentes branches doivent être donnés dans un ordre croissant des valeurs :

   ……

5) Les valeurs de cas peuvent être données par des variables :

   ……

6) Si une branche comporte plusieurs instructions, il faut les mettre entre { } :

   ……

7) Toutes les valeurs possibles de l'expression doivent être prévues dans les différentes branches du **switch** :

   ……

8) Sans instruction particulière, après le traitement d'une branche, on passe à la branche suivante :

   ……

9) Deux branches différentes peuvent comporter la même valeur de cas :

……

10) Une instruction **switch** peut toujours remplacer une instruction **if** :

……

11) Une instruction **if** peut toujours remplacer une instruction **switch** :

……

Exercice 3.14 Instruction switch (2)

Soit le programme suivant :

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

int main() {

   int n;

   cout << "Donnez un entier : ";
   cin >> n;

   switch (n) {
      case 0:  cout << "A\n";
      case 1:
      case 2:  cout << "B\n";
               break;
      case 3:
      case 4:
      case 5:  cout << "C\n";
      default: cout << "D\n";
   }

   return EXIT_SUCCESS;
}
```

Que va afficher le programme ci-dessus lorsque l'utilisateur entre comme valeur :

1) 0
2) 1
3) 4
4) 6
5) -1

Exercice 3.15 Nombre de jours dans un mois donné (2)

Même énoncé que l'exercice 3.8, mais à résoudre cette fois en utilisant l'instruction *switch* et en définissant une énumération fortement typée pour représenter les mois.

Exercice 3.16 Tracing manuel de variables

Soit l'extrait de code suivant :

```
int n = 1789;
int somme = 0;
int chiffre;
while (n > 0) {
   chiffre = n % 10;
   somme = somme + chiffre;
   n = n / 10;
}
cout << somme << endl;
```

Reporter dans la colonne correspondante du tableau ci-dessous, chaque changement de valeur des diverses variables.

n	somme	chiffre	output

Exercice 3.17 Boucle while (1)

Que va afficher à l'exécution chacun des groupes d'instructions ci-dessous ?

1)
```
int i = 0;
while (i - 10) {
   i += 2; cout << i << " ";
}
```

2)
```
int i = 0;
while (i - 10)
   i += 2; cout << i << " ";
```

3)
```
int i = 0;
while (i < 11) {
   i += 2; cout << i << " ";
}
```

4)
```
int i = 11;
while (i--) {
   cout << i-- << " ";
}
```

5)
```
int i = 12;
while (i--) {
   cout << --i << " ";
}
```

6)
```
int i = 0;
while (i++ < 10) {
   cout << i-- << " ";
}
```

7)
```
int i = 1;
while (i <= 5) {
   cout << 2 * i++ << " ";
}
```

8)
```
int i = 1;
while (i != 9) {
   cout << (i = i + 2) << " ";
}
```

Exercice 3.18 Boucle while (2)

Que va afficher le programme suivant ?

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

int main() {

   int i, j;

   i = 0;
   while (i <= 5) i++;
   cout << "A : i = " << i << endl;

   i = j = 0;
   while (i <= 5) i += j++;
   cout << "B : i = " << i << " j = " << j << endl;

   i = j = 0;
   while (i <= 5) i += ++j;
   cout << "C : i = " << i << " j = " << j << endl;

   i = j = 0;
   while (j <= 5) i += j++;
   cout << "D : i = " << i << " j = " << j << endl;

   i = j = 0;
   while (j <= 5) i += ++j;
   cout << "E : i = " << i << " j = " << j << endl;

   i = j = 0;
   while (i <= 5) i += 2; j++;
   cout << "F : i = " << i << " j = " << j << endl;

   return EXIT_SUCCESS;
}
```

Exercice 3.19 Doublement d'un avoir bancaire

Ecrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir un montant et un taux d'intérêt annuel (en %), détermine / affiche combien d'années (entières) sont nécessaires pour doubler ledit montant.

Résoudre le problème ci-dessus :
1) En utilisant une boucle while
2) Sans utiliser de boucle

IMPORTANT
- Les saisies utilisateur sont supposées correctes
- Le taux d'intérêt annuel est supposé rester constant au cours du temps

Exercice 3.20 Affichage de nombres par lots

Ecrire un programme C++ qui affiche les nombres de 20 à 1 par lots de trois comme suit :

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

Résoudre le problème de deux manières différentes :
1) En utilisant une boucle *while*
2) En utilisant une boucle *for*

Indication
- Utiliser le manipulateur *setw* défini dans la librairie *iomanip* pour réaliser l'affichage demandé

Exercice 3.21 Boucles for (1)

Que va afficher à l'exécution chacun des groupes d'instructions ci-dessous ?

1) `for (int i = 1; i < 10; ++i) {cout << i << " ";}`

2) `for (int i = 1; i < 10; i += 2) {cout << i << " ";}`

3) `for (int i = 10; i > 1; --i) {cout << i << " ";}`

4) `for (int i = 0; i < 10; ++i) {cout << i << " ";}`

5) `for (int i = 1; i < 10; i = i * 2) {cout << i << " ";}`

6) `for (int i = 1; i < 10; ++i) {if (i % 2 == 0) {cout << i << " ";}}`

Exercice 3.22 Boucles for (2)

Pour chacun des cas ci-dessous, indiquer combien de fois la boucle est exécutée :
(**NB** Dans chacun des cas, la variable de boucle *i* est supposée être non modifiée dans le corps de la boucle)

1) `for (int i = 1; i <= 10; ++i)…`

2) `for (int i = 0; i < 10; ++i)…`

3) `for (int i = 10; i > 0; --i)…`

4) `for (int i = -10; i <= 10; ++i)…`

5) `for (int i = 10; i >= 0; ++i)…`

6) `for (int i = -10; i <= 10; i = i + 2)…`

7) `for (int i = -10; i <= 10; i = i + 3)…`

Exercice 3.23 Boucles for (3)

Que va afficher à l'exécution chacun des groupes d'instructions ci-dessous ?

1)
```
int s = 1;
for (int n = 1; n <= 5; ++n) {
   s = s + n;
   cout << s << " ";
}
```

2)
```
int s = 1;
for (int n = 1; n <= 10; cout << s << " ") {
   n = n + 2;
   s = s + n;
}
```

3)
```
int s = 1;
int n;
for (n = 1; n <= 5; ++n) {
   s = s + n;
   n++;
}
cout << s << " " << n;
```

Exercice 3.24 Balle qui rebondit

Lorsqu'une balle tombe d'une hauteur initiale $h0$, sa vitesse d'impact au sol est $v0 = \sqrt{2gh0}$ (où g = 9.81 m/s^2 = constante de gravité terrestre). Immédiatement après le rebond, sa vitesse est $v1 = \varepsilon \cdot v0$ (où ε est le coefficient de rebond de la balle). Elle remonte alors à la hauteur $h = v1^2 / 2g$.

Ecrire un programme C++ qui calcule et affiche la hauteur à laquelle la balle remonte après un nombre donné de rebonds.

Les données suivantes seront entrées par l'utilisateur et le programme devra vérifier la validité partielle de ces données (partielle… car on supposera que l'utilisateur entre bien une valeur numérique) :

- ε, le coefficient de rebond, tel que $0 \le \varepsilon < 1$;
- $h0$, la hauteur initiale, telle que $h0 \ge 0$;
- n, le nombre de rebonds, tel que $n \ge 0$.

IMPORTANT

- Pour la vérification partielle des saisies utilisateur, utiliser des boucles *do…while*.
- Cherchez à résoudre le problème, non pas du point de vue formel (équations), mais par simulation du système physique (balle).
- Afficher le résultat avec 2 chiffres après la virgule.

Exercice 3.25 Boucles for imbriquées

Que va afficher à l'exécution chacun des groupes d'instructions ci-dessous ?

1)
```
for (int i = 1; i <= 3; ++i) {
   for (int j = 1; j <= 4; ++j) {
      cout << "*";
   }
   cout << endl;
}
```

2)
```
for (int i = 0; i < 4; ++i) {
   for (int j = 0; j < 3; ++j) {
      cout << "*";
   }
   cout << endl;
}
```

3)
```
for (int i = 1; i <= 4; ++i) {
   for (int j = 1; j <= i; ++j) {
      cout << "*";
   }
   cout << endl;
}
```

4)
```
for (int i = 1; i <= 3; ++i) {
   for (int j = 1; j <= 5; ++j) {
      if (j % 2 == 0) {
         cout << "*";
      } else {
         cout << "_";
      }
   }
   cout << endl;
}
```

5)
```
for (int i = 1; i <= 3; ++i) {
   for (int j = 1; j <= 5; ++j) {
      if ( (i + j) % 2 == 0) {
         cout << "*";
      } else {
         cout << " ";
      }
   }
   cout << endl;
}
```

Exercice 3.26 Triangle d'étoiles

Ecrire un programme C++ qui affiche un triangle comme dans l'exemple ci-dessous.

La hauteur du triangle (càd le nombre de lignes) est fixée par l'utilisateur.

Faire en sorte :

- que l'utilisateur soit invité à refaire sa saisie s'il entre une hauteur négative
- que la dernière ligne du triangle s'affiche sur le bord gauche de l'écran

Exemple d'exécution

```
Hauteur du triangle (h >= 0) : 10

          *
         ***
        *****
       *******
      *********
     ***********
    *************
   ***************
  *****************
*******************
```

Exercice 3.27 Le code est-il simplifiable ?

Soit le code suivant :

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

int main() {

   const int N = 5;

   int compteur = 0;

   for (int i = 0; i < N; ++i) {
      for (int j = i; j < N; ++j) {
         compteur++;
      }
   }
   cout << "compteur = " << compteur << endl;


   return EXIT_SUCCESS;
}
```

1) Que va afficher le code ci-dessus ?
2) Le code ci-dessus est-il simplifiable ? Si oui, proposer un (des) correctif(s).

Exercice 3.28 Série harmonique

Ecrire un programme C++ permettant de calculer la somme des n premiers termes de la série harmonique, c'est-à-dire :

$$s = \sum_{k=1}^{n} \frac{1}{k}$$

La valeur de n sera fixée par l'utilisateur.

Faire en sorte que l'utilisateur soit invité à refaire sa saisie s'il entre une valeur négative ou nulle (on suppose, par contre, que l'utilisateur a bien saisi un nombre et pas autre chose).

Exemple d'exécution

```
Combien de termes voulez-vous ? 0
Erreur. La valeur saisie doit etre > 0
Combien de termes voulez-vous ? 3
La somme des 3 premiers termes de la serie vaut 1.83333
```

Exercice 3.29 ppmc

Ecrire un programme C++ permettant de calculer le ppmc (plus petit multiple commun) de deux nombres entiers (> 0) *m* et *n* saisis (sur la même ligne) par l'utilisateur.

Faire en sorte que la saisie utilisateur soit vérifiée. On suppose toutefois que l'utilisateur entre des valeurs du bon type et dans l'intervalle de définition du type.

Exercice 3.30 Simulations de boucles

1) Soit la boucle `for` suivante :

```
for (; i < 10; ++i) {cout << i << endl;}
```

Récrire la boucle `for` ci-dessus, en utilisant :

a) une boucle `while`

b) une boucle `do… while`

2) Soit la boucle `while` suivante :

```
while (i-- > 10) {cout << i << endl;}
```

Récrire la boucle `while` ci-dessus, en utilisant :

a) une boucle `for`

b) une boucle `do… while`

Exercice 3.31 continue et break dans boucle do… while

Que va afficher le programme suivant ?

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

int main() {

   int n = 0;

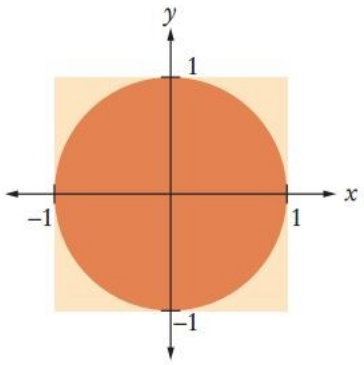
   do {
      if (n % 2 == 0) {cout << n << " est pair\n";
                       n += 3;
                       continue;
                      }
      if (n % 3 == 0) {cout << n << " est multiple de 3\n";
                       n += 5;
                      }
      if (n % 5 == 0) {cout << n << " est multiple de 5\n";
                       break;
                      }
      n++;
   } while (true);

   return EXIT_SUCCESS;
}
```

Exercice 3.32 Approximation de π par la méthode Monte-Carlo

Wikipédia *Le terme **méthode de Monte-Carlo**, ou **méthode Monte-Carlo**, désigne une famille de méthodes algorithmiques visant à calculer une valeur numérique approchée en utilisant des procédés aléatoires, c'est-à-dire des techniques probabilistes. Le nom de ces méthodes, qui fait allusion aux jeux de hasard pratiqués à Monte-Carlo, a été inventé en 1947 par Nichols Metropolis, et publié pour la première fois en 1949 dans un article coécrit avec Stanislaw Ulam.*

Le but de cet exercice est de réussir à établir une bonne approximation de π en réalisant une simulation basée sur la méthode Monte-Carlo.
Pour ce faire, on considère une cible carrée dans laquelle est inscrit un cercle de rayon R = 1 :



L'idée consiste à simuler le tir aléatoire (au hasard) de *n* fléchettes sur une telle cible. On admet que toute fléchette tirée atteint la cible. Pour simuler aléatoirement le point d'impact d'une fléchette, c'est simple : il suffit de générer aléatoirement une coordonnée x_{impact} et une coordonnée y_{impact}, comprises toutes deux dans l'intervalle [-1.0; 1.0].
Désignons par *m* le nombre de fléchettes tirées dont le point d'impact figure dans ou sur le bord du cercle.

Etant donné le caractère totalement aléatoire de nos tirs, on s'attend à ce que, pour un nombre de tirs suffisamment grand :

$$\frac{surface\ cercle}{surface\ carrée} \approx \frac{m}{n}$$

et ainsi de pouvoir en déduire une bonne approximation de π.

A vous de jouer maintenant !

Indication / prescription

- Pour réaliser des tirages aléatoires, utiliser ici la fonction *rand()* de *cstdlib*.
- Afficher le résultat avec 2 chiffres après la virgule

Exercice 3.33 Le problème des 3 portes

Wikipédia ***Le problème des 3 portes ou problème de Monty Hall*** *est un casse-tête probabiliste librement inspiré du jeu télévisé américain Let's Make a Deal. Il est simple dans son énoncé mais non intuitif dans sa résolution et c'est pourquoi on parle parfois à son sujet de* ***paradoxe de Monty Hall****. Il porte le nom de celui qui a présenté ce jeu aux États-Unis pendant treize ans, Monty Hall.*

Enoncé du problème

Le jeu oppose un présentateur à un candidat (le joueur). Le joueur est placé devant trois portes fermées. Derrière l'une d'elles se trouve une voiture (ou tout autre prix magnifique) et derrière chacune des deux autres se trouve une chèvre (ou tout autre prix sans importance). Le joueur doit tout d'abord désigner une porte. Puis le présentateur doit ouvrir une porte qui n'est ni celle choisie par le candidat, ni celle cachant la voiture (le présentateur sait quelle est la bonne porte dès le début). Le candidat a alors le droit ou bien d'ouvrir la porte qu'il a choisie initialement, ou bien d'ouvrir la troisième porte.

Les questions qui se posent au candidat sont :

1. *Que doit-il faire ?*
2. *Quelles sont ses chances de gagner la voiture en agissant au mieux ?*

Tentez de répondre aux deux questions ci-dessus, en imaginant une simulation Monte-Carlo du jeu.

Chapitre 4 : Fonctions

Exercice 4.1 Passage par valeur

Déterminer "à la main" ce que va afficher, à l'exécution, le programme ci-dessous.

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

void f(int n) {
   n *= n + 1;
   cout << "B : n = " << n << endl;
}

int main() {

   int n = 2;

   cout << "A : n = " << n << endl;
   f(n);
   cout << "C : n = " << 2 * n << endl;

   return EXIT_SUCCESS;
}
```

Exercice 4.2 Appels de fonctions

Déterminer "à la main" ce que va afficher, à l'exécution, le programme ci-dessous.

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

double k(double x) {return 2 * (x + 1);}
double h(double x) {return x * x + k(x) - 1;}
double g(double x) {return 4 * h(x);}
double f(double x) {return g(x) + sqrt(h(x));}

int main() {

   double x1 = f(2),
          x2 = g(h(2)),
          x3 = k(g(2) + h(2)),
          x4 = f(0) + f(1) + f(2),
          x5 = f(-1) + g(-1) + h(-1) + k(-1);

   cout << "x1 = " << x1 << endl;
   cout << "x2 = " << x2 << endl;
   cout << "x3 = " << x3 << endl;
   cout << "x4 = " << x4 << endl;
   cout << "x5 = " << x5 << endl;

   return EXIT_SUCCESS;
}
```

Exercice 4.3 Volume d'une pyramide à base rectangulaire

Ecrire un programme C++ mettant à disposition une fonction permettant de calculer le volume d'une pyramide à base rectangulaire.

Appliquer la fonction pour calculer le volume des 2 pyramides suivantes :

1) Pyramide 1 : base de longueur 10 et de largeur 3.5; hauteur = 12
2) Pyramide 2 : base de longueur 3.6 et de largeur 2.4; hauteur = 2.7

Afficher les résultats avec un chiffre après la virgule.

Exercice 4.4 Année bissextile

Ecrire un programme C++ affichant à l'écran si les années 1900, 2000, 2010 et 2020 sont bissextiles ou non.

Pour ce faire, écrire une fonction booléenne `estBissextile` prenant en paramètre l'année à tester.

Rappel Une année est bissextile si elle est divisible par 400 ou alors par 4 mais pas par 100.

Exercice 4.5 Affichage de caractères compris entre 2 bornes

Ecrire une fonction qui a 2 paramètres de type `unsigned char` et qui affiche, sur une même ligne, tous les caractères compris entre le premier et le deuxième paramètre (bornes comprises), pour autant que le premier représente un caractère qui précède le deuxième.

La fonction retourne une valeur booléenne indiquant si oui ou non il y a eu affichage de caractères.

Exercice 4.6 Correction d'erreurs

Quelles modifications faut-il apporter au programme ci-dessous pour le rendre correct ?

NB Aucune ligne de code ne doit être ajoutée !

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

int main() {
   f;
   g(1, 2.0);
   h(64, 65);
   return EXIT_SUCCESS;
}

void f {
   cout << "Appel de f\n";
}

void g(int i, j) {
   cout << "Appel de g avec i = " << i << " et j = " << j << "\n";
}

void h(int i; char c) {
   cout << "Appel de h avec i = " << i << " et c = '" << c << "'\n";
}
```

Exercice 4.7 Mystère, mystère !

Déterminer "à la main" ce que va afficher, à l'exécution, le programme ci-dessous.

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

void mystere(int& a, int& b);

int main() {

   int x = 3,
       y = 4;

   mystere(x, y);
   cout << "x = " << x << " " << "y = " << y << endl;

   return EXIT_SUCCESS;
}

void mystere(int& a, int& b) {
   a = a - b;
   b = b + a;
   a = b - a;
}
```

Exercice 4.8 Permutation circulaire droite de 3 réels

Ecrire un programme C++ qui met à disposition une fonction sans valeur de retour et prenant 3 paramètres de type `double` permettant d'effectuer une permutation circulaire droite de ces 3 valeurs.

Exercice 4.9 Retrait d'argent

Ecrire un programme C++ mettant à disposition une fonction permettant de retirer un certain montant sur un compte bancaire.

La fonction :

- doit prendre 2 paramètres : le solde courant du compte et le montant du retrait (que l'on supposera tous deux ≥ 0)
- doit vérifier les contraintes métier suivantes :
  - le solde courant ne peut jamais être négatif
  - si le solde courant est insuffisant pour retirer l'intégralité du montant souhaité, seul le montant maximal possible est retiré
- doit retourner le montant du retrait effectif réalisé

Vérifier le bon fonctionnement de la fonction en appliquant le scénario de test suivant :

- Solde initial du compte = 500
- Premier retrait = 300
- Afficher le montant du retrait effectif et le solde courant du compte
- Deuxième retrait = 300
- Afficher le montant du retrait effectif et le solde courant du compte

Exercice 4.10 Fonction "Opérations arithmétiques"

Ecrire une fonction qui reçoit en paramètres d'entrée 2 nombres réels (de type `double`) et un caractère, et qui fournit en paramètre de sortie le résultat correspondant à l'une des 4 opérations appliquées à ses 2 premiers paramètres, en fonction de la valeur du troisième paramètre, à savoir : addition pour le caractère '+', soustraction pour '-', multiplication pour '*' et division pour '/'. Si l'un de ces 4 caractères est passé en argument, la fonction doit renvoyer `true` comme valeur (statut) de retour; dans le cas contraire, elle doit renvoyer `false`. Ne pas tenir compte des risques de division par zéro.

Ecrire aussi un petit programme (`main`) utilisant cette fonction pour effectuer les 4 opérations sur les 2 nombres fournis en entrée par l'utilisateur. Ne pas tenir compte des éventuelles erreurs de saisie utilisateur.

Exercice 4.11 Passage par valeur vs par référence

Le programme ci-dessous contient plusieurs erreurs.

1) Pour chaque ligne fautive, indiquez son numéro et précisez la nature de l'erreur
2) En admettant maintenant que les lignes fautives ont été supprimées du programme ci-dessous, déterminer "à la main" ce qu'il va afficher à l'exécution ?

```
 1 #include <cstdlib>
 2 #include <iostream>
 3 using namespace std;
 4
 5 void f(double x);
 6 void g(double& x);
 7 void h(const double& x);
 8
 9 int main() {
10
11    const int C = 1;
12    int n = C;
13    double x = C;
14
15    f(C);
16    f(n);
17    f(x);
18    g(C);
19    g(n);
20    g(x);
21    h(C);
22    h(n);
23    h(x);
24
25    return EXIT_SUCCESS;
26 }
27
28 void f (double x) {
29    x = x + 1;
30    cout << "x = " << x << endl;
31 }
32
33 void g (double& x) {
34    x = x + 1;
35    cout << "x = " << x << endl;
36 }
37
38 void h (const double& x) {
39    x = x + 1;
40    cout << "x = " << x << endl;
41 }
```

Exercice 4.12 Portée des variables

Déterminer "à la main" ce que va afficher, à l'exécution, le programme ci-dessous.

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

int a, b;

int f(int c) {
   int n = 0;
   a = c;
   if (n < c) {
      n = a + b;
   }
   return n;
}

int g(int c) {
   int n = 0;
   int a = c;
   if (n < f(c)) {
      n = a + b;
   }
   return n;
}

int main() {
   int i = 1;
   int b = g(i);
   cout << a + b + i << endl;
   return EXIT_SUCCESS;
}
```

Exercice 4.13 Fonction "compteur"

Ecrire un programme C++ qui appelle 3 fois la même fonction `afficher`, cette dernière se contentant d'afficher, à chaque appel, le nombre total de fois où elle a été appelée, sous la forme :

```
Appel numéro n
```

On supposera que la fonction `afficher` est une fonction sans argument ni valeur de retour.

Exercice 4.14 Surcharge (1)

Soient les déclarations suivantes :

```
void f(char);  // fonction I
void f(int);   // fonction II
void f(float); // fonction III

bool b = true;
char c = 'A';
short s = 1;
int i = 2;
float x = 3.f;
double y = 4.0;
```

Les appels suivants sont-ils corrects et, si oui, quelles seront les fonctions effectivement appelées et les conversions éventuelles mises en œuvre ?

1) `f(b);`
2) `f(c);`
3) `f(s);`
4) `f(i);`
5) `f(x);`
6) `f(y);`

Exercice 4.15 Surcharge (2)

Soient les déclarations suivantes :

```
void f(int, float); // fonction I
void f(float, int); // fonction II

int i = 1;
char c = 'A';
float x = 2.f;
double y = 3.0;
```

Les appels suivants sont-ils corrects et, si oui, quelles seront les fonctions effectivement appelées et les conversions éventuelles mises en œuvre ?

1) `f(i, c);`
2) `f(c, i);`
3) `f(i, i);`
4) `f(x, c);`
5) `f(c, x);`
6) `f(x, x);`
7) `f(i, y);`
8) `f(y, i);`
9) `f(y, y);`

Exercice 4.16 Suite de Syracuse

Partie A

La suite de Syracuse repose sur un principe simple. Prenez un nombre entier positif quelconque :

- s'il est pair, divisez-le par 2;
- s'il est impair, multipliez-le par 3 et ajoutez 1.

Renouvellez cette opération plusieurs fois. Après suffisamment d'itérations, vous devriez finir par tomber sur la valeur 1.
Par exemple, à partir de 17, on trouve la suite de valeurs : 52 26 13 40 20 10 5 16 8 4 2 1.

On demande ici d'écrire un programme C++ (dans un seul fichier) qui demande à l'utilisateur d'entrer un nombre entier n ($1 \leq n \leq 10000$) et qui affiche à l'écran 1°) les valeurs successives de la suite de Syracuse relative à ce nombre (en s'arrêtant bien sûr à 1), 2°) combien d'itérations ont été nécessaires pour parvenir à 1.

Partie B

Modifiez le programme précédent pour qu'il affiche le nombre d'itérations nécessaires pour parvenir à 1, non plus pour le nombre entré par l'utilisateur, mais pour les n premiers entiers positifs (n fixé par l'utilisateur, $1 \leq n \leq 1000$)

Prescriptions (valables pour les parties A et B) :

- Structurez votre code en sous-programmes
- Il n'est pas demandé de vérifier les saisies utilisateurs

Exemple d'exécution (partie A)

```
Entrez un entier dans l'intervalle [1, 10000] : 29
Suite de Syracuse pour n = 29 :
88
44
22
11
34
17
52
26
13
40
20
10
5
16
8
4
2
1
Suite terminee en 18 iterations.
```

Exemple d'exécution (partie B)

```
Entrez un entier dans l'intervalle [1, 1000] : 10
Nbre d'iter. pour n = 1    :   0
Nbre d'iter. pour n = 2    :   1
Nbre d'iter. pour n = 3    :   7
Nbre d'iter. pour n = 4    :   2
Nbre d'iter. pour n = 5    :   5
Nbre d'iter. pour n = 6    :   8
Nbre d'iter. pour n = 7    :  16
Nbre d'iter. pour n = 8    :   3
Nbre d'iter. pour n = 9    :  19
Nbre d'iter. pour n = 10   :   6
```

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];
```

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

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]; }`

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]

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

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.

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.

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.

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.

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.

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.

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.

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.

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

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.

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]
```

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]
```

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
```

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.

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
```

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

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)

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*>.

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.

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*>.

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]
```

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
```

Chapitre 6 : Chaînes de caractères

Exercice 6.1 Analyse d'un caractère

Ecrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir un caractère, détermine / affiche à l'écran si ce caractère :

1°) est une lettre de l'alphabet
2°) est une lettre minuscule de l'alphabet
3°) est un chiffre
4°) est un symbole de ponctuation

Le résultat doit être affiché comme dans l'exemple d'exécution suivant :

```
Entrez un caractere : a

Code ASCII de 'a'                  : 97
'a' est une lettre de l'alphabet   : vrai
'a' est une lettre minuscule       : vrai
'a' est un chiffre                 : faux
'a' est un caractere de ponctuation : faux
```

Exercice 6.2 Constructeurs

Indiquer ce que va afficher[1] chacune des séquences d'instructions suivantes :

[1] Il se peut qu'une séquence provoque une erreur à la compilation ou à l'exécution, voire ait un comportement indéfini. Le cas échéant, répondez "Erreur à la compilation", "Erreur à l'exécution" ou "Comportement indéfini"

1.
```
string s1 = "ABC";
string s2(s1, 1);
cout << '|' << s2 << '|' << endl;
```

2.
```
string s1 = "ABC";
string s2(s1, 2, 3);
cout << '|' << s2 << '|' << endl;
```

3.
```
string s1 = "ABC";
string s2(s1, 3, 2);
cout << '|' << s2 << '|' << endl;
```

4.
```
string s1 = "ABC";
string s2(s1, 4, 1);
cout << '|' << s2 << '|' << endl;
```

5.
```
string s("ABC", 3);
cout << '|' << s << '|' << endl;
```

6.
```
string s("ABC", 4);
cout << '|' << s << '|' << endl;
```

7.
```
string s("ABC", 5);
cout << '|' << s << '|' << endl;
```

8.
```
string s(3, 'A');
cout << '|' << s << '|' << endl;
```

Exercice 6.3 assign, '+', '+=' et append

Indiquer ce que va afficher[1] chacune des séquences d'instructions suivantes :

[1] *Il se peut qu'une séquence provoque une erreur à la compilation ou à l'exécution, voire ait un comportement indéfini. Le cas échéant, répondez "Erreur à la compilation", "Erreur à l'exécution" ou "Comportement indéfini"*

1. ```
   string s1 = "ABC";
   string s2;
   s2.assign(s1, 2, 3);
   cout << '|' << s2 << '|' << endl;
   ```

2. ```
   string s1 = "ABC";
   string s2;
   s2.assign(s1, 3, 2);
   cout << '|' << s2 << '|' << endl;
   ```

3. ```
   string s;
   s.assign("ABC", 2);
   cout << '|' << s << '|' << endl;
   ```

4. ```
   string s;
   s.assign(2, 65);
   cout << '|' << s << '|' << endl;
   ```

5. ```
   string s;
   s.assign(4, '\101');
   cout << '|' << s << '|' << endl;
   ```

6. ```
   string s;
   s.assign(4, '\x42');
   cout << '|' << s << '|' << endl;
   ```

7. ```
   string s = string("A") + string("BC");
   cout << '|' << s << '|' << endl;
   ```

8. ```
   string s = string("A") + "BC";
   cout << '|' << s << '|' << endl;
   ```

9. ```
   string s = 'A' + string("BC");
   cout << '|' << s << '|' << endl;
   ```

10. ```
    string s = "A" + "BC";
    cout << '|' << s << '|' << endl;
    ```

11. ```
    string s = "AB";
    s += 'C';
    cout << '|' << s << '|' << endl;
    ```
    ```

12.
```
string s1 = "";
string s2 = "123";
s1.append(s2);
cout << '|' << s1 << '|' << endl;
```

13.
```
string s1 = "";
string s2 = "123";
s1.append(s2, 1, 3);
cout << '|' << s1 << '|' << endl;
```

14.
```
string s;
s.append("123", 2);
cout << '|' << s << '|' << endl;
```

15.
```
string s;
s.append(2, '2');
cout << '|' << s << '|' << endl;
```

Exercice 6.4 Affichage d'un petit dessin

Sans utiliser le concept de boucles, écrire, le plus judicieusement possible, un programme C++ permettant d'afficher à l'écran le dessin suivant :

```
   *
  ***
 *****
*******
 *****
  ***
   *
```

Exercice 6.5 Vérifier qu'une chaîne de caractères est vide

On souhaite tester si une chaîne de caractères *str* de type string est vide ou non.
Proposer 4 manières différentes d'écrire le test.

Exercice 6.6 [ ], at, length, size, resize et substr

Indiquer ce que va afficher[1] chacune des séquences d'instructions suivantes :

[1] Il se peut qu'une séquence provoque une erreur à la compilation ou à l'exécution, voire ait un comportement indéfini. Le cas échéant, répondez "Erreur à la compilation", "Erreur à l'exécution" ou "Comportement indéfini"

1.
```
string s = "ABC";
cout << '|' << s[1] << '|' << endl;
```

2.
```
string s1 = "ABC";
string s2 = s1[1];
cout << '|' << s2 << '|' << endl;
```

3.
```
string s1 = "ABC";
string s2;
s2 = s1[1];
cout << '|' << s2 << '|' << endl;
```

4.
```
string s = "ABC";
cout << '|' << s[3] << '|' << endl;
```

5.
```
string s = "ABC";
cout << '|' << s[4] << '|' << endl;
```

6.
```
string s = "ABC";
cout << '|' << s.at(3) << '|' << endl;
```

7.
```
string s;
cout << s.length() << endl;
```

8.
```
string s = "ABC";
cout << s.length() << endl;
```

9.
```
string s = "ABC";
cout << s.size() << endl;
```

10.
```
string s = "ABC";
s.resize(5);
cout << '|' << s << '|' << endl;
```

11.
```
string s = "ABC";
s.resize(2, 'x');
cout << '|' << s << '|' << endl;
```

12.
```
string s = "ABC";
s.resize(4, 'x');
cout << '|' << s << '|' << endl;
```

13.
```
string s = "ABCDE";
cout << '|' << s.substr(1, 2) << '|' << endl;
```

14.
```
string s = "ABCDE";
cout << '|' << s.substr(0, 10) << '|' << endl;
```

15.
```
string s = "ABCDE";
cout << '|' << s.substr(3) << '|' << endl;
```

16.
```
string s = "ABCDE";
cout << '|' << s.substr() << '|' << endl;
```

Exercice 6.7 Milieu d'une chaîne de caractères

Ecrire une fonction C++ qui prend en paramètre une chaîne de caractères *str* (de type string) et qui retourne une chaîne de caractères (de type string) contenant :

- le caractère médian de *str*, si *str* comporte un nombre impair de caractères
- les 2 caractères médians de *str*, si *str* comporte un nombre pair de caractères

Exemples milieu("ABC") renvoie "B"; milieu("ABCD") renvoie "BC"

Exercice 6.8 Inversion récursive d'une chaîne de caractères

Ecrire une fonction récursive :

```
void inverser(string& str)
```

permettant d'inverser le contenu de la chaîne de caractères *str* passée en paramètre.

Ecrire également un petit programme de test permettant de vérifier le bon fonctionnement de la fonction *inverser*.

Exercice 6.9 Numération romaine

Wikipédia *La **numération romaine** est un système de numération additive utilisé par les Romains de l'Antiquité. Les chiffres romains sont représentés à l'aide de symboles combinés entre eux, notamment par les signes I, V, X, L, C, D et M, représentant respectivement les nombres 1, 5, 10, 50, 100, 500 et 1000.*
Un nombre écrit en chiffres romains se lit de gauche à droite. En première approximation, sa valeur se détermine en faisant la somme des valeurs individuelles de chaque symbole, sauf quand l'un des symboles précède un symbole de valeur supérieure; dans ce cas, on soustrait la valeur du premier symbole au deuxième.
Exemple : XIV = X + IV = 10 + (5 - 1) = 14

En s'inspirant de ce qui précède, on demande ici d'écrire une fonction (non récursive) permettant de convertir un nombre romain (supposé correct) en entier décimal.
Ecrire également un petit programme de test (*main*) permettant de vérifier le bon fonctionnement de votre fonction de conversion.

Exercice 6.10 Analyse d'un nombre entier (2)

Sans utiliser la librairie *cmath*, ni la librairie *sstream*, ni le concept de boucle mais en utilisant exclusivement les services offerts par la classe *string*, écrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir un nombre entier (de type *int*) ≥ 0, affiche à l'écran combien de chiffres contient ce nombre et ce que valent le premier chiffre et le dernier chiffre du nombre.

NB On supposera ici la saisie utilisateur correcte.

Exemple d'exécution

```
Entrez un nombre entier >= 0 : 123

Nombre saisi       : 123
Nombre de chiffres : 3
Premier chiffre    : 1
Dernier chiffre    : 3
```

Exercice 6.11 Prénom, nom et acronyme (1)

Ecrire un programme C++ qui après avoir demandé à l'utilisateur de saisir son prénom et son nom (sur la même ligne), affiche à l'écran son prénom, son nom ainsi que son acronyme.

NB L'acronyme s'obtient en concaténant la première lettre du prénom, la première lettre du nom et la dernière lettre du nom, le tout mis entièrement en majuscules.

IMPORTANT
On supposera :
- que l'utilisateur n'entre qu'un seul prénom et qu'un seul nom
- que le nom ne contient aucun espace blanc

Il n'est pas demandé ici de tenir compte des cas où l'utilisateur n'entrerait que son prénom, que son nom, voire ni son prénom, ni son nom.

Exemple d'exécution

```
Entrez votre prenom et votre nom : Alexandre Dumas

Votre prenom est   : Alexandre
Votre nom est      : Dumas
Votre acronyme est : ADS
```

Exercice 6.12 insert, replace et erase

Indiquer ce que va afficher[1] chacune des séquences d'instructions suivantes :

[1] Il se peut qu'une séquence provoque une erreur à la compilation ou à l'exécution, voire ait un comportement indéfini. Le cas échéant, répondez "Erreur à la compilation", "Erreur à l'exécution" ou "Comportement indéfini"

1.
```
string s1 = "ABC";
string s2 = "123";
s1.insert(1, s2);
cout << '|' << s1 << '|' << endl;
```

2.
```
string s1 = "ABC";
string s2 = "123";
s1.insert(4, s2);
cout << '|' << s1 << '|' << endl;
```

3.
```
string s1 = "ABC";
string s2 = "123";
s1.insert(2, s2, 1, 2);
cout << '|' << s1 << '|' << endl;
```

4.
```
string s1 = "ABC";
string s2 = "123";
s1.insert(2, s2, 0, string::npos);
cout << '|' << s1 << '|' << endl;
```

5.
```
string s = "ABC";
s.insert(3, "123", 2);
cout << '|' << s << '|' << endl;
```

6.
```
string s = "ABC";
s.insert(1, 2, '3');
cout << '|' << s << '|' << endl;
```

7.
```
string s1 = "ABCDEF";
string s2 = "123";
s1.replace(1, 2, s2);
cout << '|' << s1 << '|' << endl;
```

8.
```
string s1 = "ABCDEF";
string s2 = "123";
s1.replace(2, 4, s2);
cout << '|' << s1 << '|' << endl;
```

9.
```
string s1 = "ABCDEF";
string s2 = "123";
s1.replace(1, 2, s2, 1, 2);
cout << '|' << s1 << '|' << endl;
```

10.
```
string s = "ABCDEF";
s.replace(2, 3, "123", 2);
cout << '|' << s << '|' << endl;
```

11.
```
string s = "ABCDEF";
s.erase();
cout << '|' << s << '|' << endl;
```

12.
```
string s = "ABCDEF";
s.erase(2);
cout << '|' << s << '|' << endl;
```

13.
```
string s = "ABCDEF";
s.erase(2, 2);
cout << '|' << s << '|' << endl;
```

Exercice 6.13 Compter le nombre d'occurrences

Ecrire une fonction qui renvoie combien de fois un caractère donné se trouve dans une chaîne de caractères (de type *string*) donnée.

Proposer deux implémentations de la fonction :

- L'une utilisant la fonction *find* de la classe *string*
- L'autre utilisant la librairie *algorithm*

Exercice 6.14 Analyse d'une adresse

Ecrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir (sur la même ligne) un nom et un numéro de rue, affiche séparément à l'écran, d'une part le nom de la rue, d'autre part le numéro de la rue.

Exemple d'exécution

```
Entrez le nom et le numero de la rue : Impasse des Acacias 24a

Le nom de la rue est : Impasse des Acacias
Le no de la rue est  : 24a
```

Exercice 6.15 Prénom, nom et acronyme (2)

Même énoncé que celui de l'exercice 6.11, à la différence importante près que, ici, il est demandé de tenir compte des cas où le nom serait constitué de plusieurs mots.

Exemples d'exécution

```
Entrez votre prenom et votre nom : Ludwig van Beethoven

Votre prenom est   : Ludwig
Votre nom est      : van Beethoven
Votre acronyme est : LVN


Entrez votre prenom et votre nom : Valery Giscard d'Estaing

Votre prenom est   : Valery
Votre nom est      : Giscard d'Estaing
Votre acronyme est : VGG
```

Exercice 6.16 Saison correspondant à une date donnée

Ecrire le plus proprement et judicieusement possible un programme C++ qui, après avoir demandé à l'utilisateur de saisir une date au format *jj.mm* (par ex 31.12), détermine / affiche la saison à laquelle correspond cette date.

Exemple d'exécution

```
Entrez une date sous la forme jj.mm (par ex 31.12) : 21.3
Le 21.03 se situe au printemps.
```

IMPORTANT

- On suppose la saisie utilisateur correcte
- On suppose les saisons définies comme suit :
  - 21.12 – 20.03 hiver
  - 21.03 – 20.06 printemps
  - 21.06 – 20.09 été
  - 21.09 – 20.12 automne

Exercice 6.17 Analyse d'un nombre entier (3)

Sans utiliser la librairie *cmath*, ni la fonction *to_string()* de la librairie *string*, ni le concept de boucle, écrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir un nombre entier (de type *int*) ≥ 0, affiche à l'écran combien de chiffres contient ce nombre et ce que valent le premier chiffre et le dernier chiffre du nombre.

NB On supposera ici la saisie utilisateur correcte.

IMPORTANT
Le nombre saisi par l'utilisateur doit obligatoirement être stocké dans une variable de type *int*

Exemple d'exécution

```
Entrez un nombre entier >= 0 : 123

Nombre saisi       : 123
Nombre de chiffres : 3
Premier chiffre    : 1
Dernier chiffre    : 3
```

Exercice 6.18 Conversions en base 8 et en base 16

Ecrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir un entier *n* (de type *int*) ≥ 0, affiche à l'écran (exactement comme dans l'exemple d'exécution ci-dessous) ce que vaut *n* en base 8 et en base 16.

NB On supposera ici la saisie utilisateur correcte.

Indication Consulter la documentation de la librairie *ios*

Exemple d'exécution

```
Entrez un nombre entier >= 0 : 255

(255)10 = (377)8 = (FF)16
```

Exercice 6.19 Saisie contrôlée d'un entier entre 2 bornes (1)

Ecrire un programme C++ qui :

1) demande à l'utilisateur de saisir un entier (de type *int*) entre deux bornes données (*min* et *max* également de type *int*)

2) affiche à l'écran la valeur saisie par l'utilisateur

IMPORTANT

- **La saisie utilisateur doit être entièrement contrôlée** (l'utilisateur doit être invité à refaire sa saisie aussi longtemps que cette dernière est incorrecte).
- Les bornes min et max sont à définir en tant que constantes dans le programme.
- Indications
  - A chaque flux est associé un "statut d'erreur" composé de 4 bits. Lorsque le flux est "sain", ces 4 bits sont tous à 0. Lorsque, par exemple, une tentative de lecture sur le flux *cin* échoue, l'un de ces 4 bits est mis à 1. Pour pouvoir continuer à utiliser le flux (pour refaire une nouvelle tentative de lecture), il s'agit au préalable de le remettre en état en remettant les 4 bits du statut d'erreur à 0. Ceci se réalise à l'aide de l'instruction : `cin.clear();`
  - Mais cela ne suffit pas. Il faut encore prendre soin de vider le buffer associé au flux afin de supprimer, notamment, le caractère responsable de l'erreur de lecture (qui, puisqu'il n'a pas pu être extrait du flux, s'y trouve toujours).
    Pour ce faire, on peut procéder de diverses manières.
    Il est demandé ici d'utiliser la méthode *ignore*.

Exemple d'exécution

```
Donnez un entier dans l'intervalle [1, 10] : a
Saisie incorrecte. Veuillez SVP recommencer.

Donnez un entier dans l'intervalle [1, 10] : 0
Saisie incorrecte. Veuillez SVP recommencer.

Donnez un entier dans l'intervalle [1, 10] : 11
Saisie incorrecte. Veuillez SVP recommencer.

Donnez un entier dans l'intervalle [1, 10] : 5
L'entier que vous avez saisi est 5.
```

Exercice 6.20 Saisie contrôlée d'un entier entre 2 bornes (2)

Même énoncé que l'exercice 6.19, à la différence près que, ici, le programme est censé non plus lire un *int* mais un *unsigned int.*

Exercice 6.21 Saisie contrôlée d'un entier entre 2 bornes (3)

Même énoncé que l'exercice 6.19, sauf qu'ici la saisie contrôlée est à implémenter en tant que fonction et que l'on souhaite mettre en œuvre les possibilités de la compilation séparée.

NB On supposera ici que la borne minimale est inférieure ou égale à la borne supérieure

Chapitre 7 : Classes

Exercice 7.1 Point (1)

En appliquant le principe d'encapsulation, implémenter une classe Point permettant de manipuler un point du plan. On prévoira :

- un constructeur recevant en argument les coordonnées (float) du point
- une fonction membre deplacer effectuant une translation définie par ses deux arguments (float)
- une fonction membre afficher se contentant d'afficher à l'écran les coordonnées cartésiennes du point sous la forme : (x,y)

On écrira séparément :

- un fichier source constituant la **déclaration** de la classe
- un fichier source correspondant à sa **définition**

Ecrire aussi un petit programme de test (main) déclarant un point, l'affichant, le déplaçant et l'affichant à nouveau.

Exercice 7.2 Point (2)

Implémenter une classe Point, analogue à celle de l'exercice 7.1, mais ne comportant pas de fonction afficher. Pour respecter le principe d'encapsulation des données, prévoir deux fonctions membre publiques (nommées abscisse et ordonnee) fournissant en retour respectivement l'abscisse et l'ordonnée d'un point.

Adapter le petit programme de test (main) de l'exercice 7.1 pour qu'il fonctionne avec cette nouvelle version de la classe Point.

Exercice 7.3 Pays

1) Implémenter une classe Pays permettant de stocker le nom, le nombre d'habitants ainsi que la superficie d'un pays.
2) A l'aide de la classe Pays, écrire un programme C++ qui, à partir d'un ensemble de pays donnés (codé "en dur"), affiche :
   - le pays ayant la plus grande superficie
   - le pays le plus peuplé
   - le pays ayant la densité de population la plus élevée

IMPORTANT
- Résoudre le problème ci-dessus en exploitant la compilation séparée

Des données réelles peuvent être trouvées facilement sur internet

Exercice 7.4 Robot

1) Implémenter une classe Robot permettant de modéliser un robot se déplaçant le long d'un axe horizontal gradué. Le robot se déplace soit vers la droite, soit vers la gauche. Initialement le robot se déplace vers la droite, mais il peut, en tout temps, faire demi-tour pour ensuite se déplacer dans la direction opposée.

   La classe doit mettre à disposition :
   - un constructeur permettant de définir la position initiale (de type int) du robot
     Important Si l'utilisateur ne fournit pas de valeur pour la position initiale du robot, on considérera que cette dernière vaut 0.
   - une fonction membre *deplacer* permettant au robot de se déplacer de n unités dans la direction courante
     Important Si l'utilisateur ne fournit pas de valeur pour n, le robot se déplacera par défaut d'une unité.
   - une fonction membre *faireDemiTour* permettant au robot de faire demi-tour
   - une méthode *getPosition* retournant la position courante du robot

2) Ecrire un petit programme de test (*main*) permettant de vérifier le bon fonctionnement de la classe Robot.

IMPORTANT
Résoudre le problème ci-dessus sans faire appel à la compilation séparée.

Exercice 7.5 Point (3)

Ajouter à la classe Point de l'exercice 7.2 (comportant un constructeur et trois fonctions membre deplacer, abscisse et ordonnee) une nouvelle fonction membre opérateur permettant de sommer deux points. On conviendra que sommer deux points revient à sommer leurs abscisses, respectivement leurs ordonnées.

Ecrire aussi un petit programme de test (main) qui, après avoir déclaré 2 points, effectue leur somme puis affiche à l'écran les coordonnées cartésiennnes du point résultant.

Exercice 7.6 Point (4)

Ajouter à la classe Point de l'exercice 7.2 (comportant un constructeur et trois fonctions membre deplacer, abscisse et ordonnee) les nouvelles fonctions membres suivantes :

- rotation qui effectue une rotation dont l'angle (exprimé en radians) est fournit en argument
- rho et theta qui fournissent en retour les **coordonnées polaires** du point

Ecrire aussi un petit programme de test (main) qui effectue 8 rotations successives de 45° chacune autour de l'origine (0,0) d'un point p ayant pour coordonnées cartésiennes initiales : (1, 0). Afficher après chaque rotation les nouvelles coordonnées cartésiennes et polaires du point p.

Prescriptions

- Pour l'angle theta, choisir une représentation entre 0 et 2π et non entre $-\pi$ et $+\pi$.
- Afficher les coordonnées cartésiennes et polaires avec 3 chiffres après la virgule

Exercice 7.7 Surcharges d'opérateurs

Soit C une classe encapsulant un entier (de type int).

Compléter la partie notée **< à compléter >** dans le code ci-après de telle sorte que, à l'exécution, celui-ci produise le résultat suivant :

```
n = 0
n = 1
false
true
n = 1
n = 2
n = 3
n = 8
n = 8
n = 12
```

```
< à compléter >

int main() {
   C c0, c1(1), c2 = 5;
   const C C3{7};

   cout << c0 << endl
        << c1 << endl;

   cout << boolalpha;
   cout << (c0 == c1) << endl;
   cout << (c0 != c1) << endl;
   cout << noboolalpha;

   cout << c1++ << endl;
   cout << c1 << endl;
   cout << ++c1 << endl;

   cout << c1 + c2 << endl;
   c1 += c2;
   cout << c1 << endl;
   cout << C3 + c2 << endl;

   return EXIT_SUCCESS;
}
```

IMPORTANT

- Le code de la fonction main ne doit pas être modifié
- Compléter la partie manquante en évitant toute redondance du code

Exercice 7.8 Membres statiques

Implémenter une classe *Objet* qui permet d'attribuer automatiquement un numéro unique à chaque nouvel objet créé (1 au premier, 2 au second…). On ne cherchera pas à réutiliser les numéros d'objets éventuellement détruits. On dotera la classe de :

- un constructeur sans paramètre
- un destructeur
- une fonction membre *no* fournissant le numéro attribué à l'objet
- une fonction membre *prochainNo* fournissant la valeur du prochain no qui sera attribué
- une fonction membre *compteur* fournissant combien d'objets existent à un instant donné

Ecrire aussi un petit programme de test (*main*) permettant de vérifier le bon fonctionnement de la classe *Objet.*

IMPORTANT
Résoudre le problème ci-dessus sans faire appel à la compilation séparée.

Exercice 7.9 Voitures

Compléter la partie notée **< *à compléter* >** dans le code ci-après de telle sorte que, à l'exécution, celui-ci produise le résultat suivant :

```
Prix de l'essence : 1.50 Frs

Prix de l'essence : 1.34 Frs

Capacite du reservoir [l]       : 52
Consommation moyenne [l/100km] : 6.7
Nb litres restants              : 52.0

Cout du trajet : 89.78 Frs

Capacite du reservoir [l]       : 52
Consommation moyenne [l/100km] : 6.7
Nb litres restants              : 37.0

Cout du trajet : 17.96 Frs

Capacite du reservoir [l]       : 52
Consommation moyenne [l/100km] : 6.7
Nb litres restants              : 23.6
```

```
< à compléter >

int main() {

   afficherPrixEssence(Voiture::getPrixEssence());

   Voiture::setPrixEssence(1.34);
   afficherPrixEssence(Voiture::getPrixEssence());

   Voiture v(52, 6.7);

   afficherVoiture(v);
   afficherCoutTrajet(v.coutTrajet(1000));
   afficherVoiture(v);
   afficherCoutTrajet(v.coutTrajet(200));
   afficherVoiture(v);

   return EXIT_SUCCESS;
}
```

IMPORTANT

- Le code de la fonction *main* ne doit pas être modifié
- Le coût du trajet ne prend en compte que les frais d'essence
- Faire l'hypothèse que le réservoir est toujours complètement vidé avant d'être complètement rempli (jamais de plein partiel)

Exercice 7.10 Messages

1) Implémenter une classe *Message* permettant de modéliser un message (email).
   Un message se caractérise par un expéditeur, un destinataire, une date-heure de création ainsi qu'un contenu (le texte du message).
   La classe doit mettre à disposition :
   - un constructeur qui prend en paramètres l'expéditeur et le destinataire et qui fixe automatiquement la date-heure courante[1]
     [1] Considérer qu'une fois celle-ci fixée, elle ne doit plus pouvoir être modifiée
   - une fonction membre qui permet d'ajouter une ligne de texte au contenu du message
   - une fonction membre *toString* qui convertit l'objet message en une unique chaîne de caractères comme suit "From : xxx\nTo : yyy\nDate : zzz\n…"
   - une fonction membre qui permet d'afficher à l'écran le message.

2) A l'aide de la classe *Message* écrire un programme C++ qui crée ("en dur") un message et affiche ce dernier à l'écran.

Exemple d'exécution

```
From : Pierre Burki
To   : Alfred Strohmeier
Date : 17.02.2016 20:01

Cher ami,
RDV demain a 9h.
Meilleures salutations.

Pierre
```

IMPORTANT

- Résoudre le problème ci-dessus en exploitant la compilation séparée
- **Indication** Pour la question de la date-heure de création et de sa mise en forme, consulter la librairie <*ctime*> et, en particulier, la fonction *strftime*

Exercice 7.11 Mailbox

1) Implémenter une classe *Mailbox* permettant de modéliser une boîte aux lettres (mailbox) contenant divers message (emails) tels que définis dans l'exercice 7.10.
   La classe doit mettre à disposition :
   - une fonction membre permettant d'ajouter un message à la mailbox
     Important La fonction membre ajoute le message à la mailbox si et seulement si ledit message ne s'y trouve pas déjà (doublon). Dans le cas contraire, elle ne fait rien.
   - une fonction membre permettant de récupérer le ième message stocké dans la mailbox
     Important Si le ième message n'existe pas, la fonction se contente de propager l'exception prédéfine *out_of_range*.
   - une fonction membre permettant de supprimer le ième message stocké dans la mailbox
     Important La fonction membre ne fait rien si la suppression n'a pas de sens (par ex si la mailbox est vide).
   - une fonction membre retournant combien de messages sont stockés dans la mailbox

2) Ecrire un petit programme de test (*main*) permettant de vérifier le bon fonctionnement de la classe *Mailbox*

Exercice 7.12 Personnes

1) Implémenter une classe *Personne* sachant que :
   - une personne a un nom, un prénom et une adresse
   - une adresse se caractérise par un nom de rue, un numéro de rue, un code postal et une localité
   - une personne a 0, 1 ou plusieurs hobbies
   - une personne a 0, 1 ou plusieurs amis

   Contraintes
   - Adresse
     - Implémenter le concept d'adresse au moyen d'une classe dédiée *Adresse*
   - Hobbies
     - Implémenter les hobbies au moyen d'un type énuméré fortement typé.
     - Les hobbies possibles sont : sport, musique, cinéma, lecture
     - Il doit être possible, en tout temps, d'ajouter un (ou plusieurs) nouvel hobby à quelqu'un
     - La non-insertion de doublons n'est pas à implémenter
     - La suppression d'un hobby n'est pas à implémenter
   - Amis
     - Un ami est une personne
     - Il doit être possible, en tout temps, d'ajouter un (ou plusieurs) nouvel ami à quelqu'un
     - La non-insertion de doublons n'est pas à implémenter
     - L'amitié est réciproque (si on ajoute un ami *A* à *B*, alors *B* doit se voir automatiquement ajouté comme ami de *A*) mais pas transitive (les amis de mes amis ne sont pas forcément mes amis)
     - La suppression d'un ami n'est pas à implémenter
   - Général
     - N'implémenter que les membres (données et fonctions) strictement nécessaires

   IMPORTANT
   Résoudre le problème ci-dessus en exploitant la compilation séparée.

2) Vérifier votre implémentation en exécutant le programme de test donné ci-après. L'exécution de ce programme doit produire le résultat suivant :

```
Paul Ecoffey
Chemin des Lilas 7A
1400 Yverdon-les-Bains
Hobbies : [musique, cinema, lecture]
Ami(e)s : [Alexandre Grandjean, Julie Ducotterd]

Alexandre Grandjean
Avenue des sports 18
1000 Lausanne
Hobbies : []
Ami(e)s : [Paul Ecoffey]

Julie Ducotterd
Rue des Acacias 21
1800 Vevey
Hobbies : [sport, cinema]
Ami(e)s : [Paul Ecoffey]
```

```
int main() {
   Adresse a1 = {"Chemin des Lilas", "7A", 1400, "Yverdon-les-Bains"},
           a2 = {"Avenue des sports", "18", 1000, "Lausanne"},
           a3 = {"Rue des Acacias", "21", 1800, "Vevey"};

   Personne p1 = {"Ecoffey", "Paul", a1, {Hobby::MUSIQUE}},
            p2 = {"Grandjean", "Alexandre", a2, {}, {&p1}},
            p3 = {"Ducotterd", "Julie", a3, {Hobby::SPORT, Hobby::CINEMA}};

   p1.ajouterHobbies({Hobby::CINEMA, Hobby::LECTURE});
   p1.ajouterAmis({&p3});

   cout << p1 << endl << endl;
   cout << p2 << endl << endl;
   cout << p3 << endl << endl;

   return EXIT_SUCCESS;
}
```

Exercice 7.13 Constructeur de (re)copie et opérateur d'affectation

En C, il n'existe pas de véritable type chaîne de caractères, mais simplement une convention de représentation des chaînes (suite de caractères terminée par un caractère de code nul). Un certain nombre de fonctions (*strcpy*, *strcat*…) utilisant cette convention permettent les manipulations classiques (copie, concaténation…).

Cet exercice vous demande de définir une classe nommée `Chaine` offrant des possibilités plus proches d'un véritable type chaîne.

Pour ce faire, on prévoira :

- comme **données membre** :
  - la longueur courante de la chaîne
  - l'adresse d'une zone allouée dynamiquement, destinée à recevoir la suite de caractères (**NB** pas nécessaire d'y ranger le caractère nul de fin, puisque la longueur de la chaîne est définie par ailleurs)

    Le contenu d'un objet de type `Chaine` pourra donc évoluer par un simple jeu de gestion dynamique.
- comme **constructeurs** :
  - `Chaine()` : initialise une chaine vide
  - `Chaine(const char*)` : initialise la chaîne avec la chaîne (au sens C) dont on fournit l'adresse en argument
  - constructeur de (re)copie
- comme **opérateurs** (on pourrait bien sûr en ajouter beaucoup d'autres !) :
  - << : pour l'affichage à l'écran d'une chaîne
  - = : pour l'affectation entre objets de type `Chaine` (penser à l'affectation multiple)

NB On trouve dans la bibliothèque standard C++ une classe `string` offrant, entre autres, les fonctionnalités évoquées ici. Pour conserver son intérêt à l'exercice, il ne faut bien sûr pas l'utiliser ici.

Chapitre 8 : Généricité

Exercice 8.1 Instanciations d'une fonction générique

Soit la définition de fonction générique suivante :

```
template <typename T, typename U> T f(T x, U y, T z) {
   return x + y + z;
}
```

et soient les déclarations de variables suivantes :

```
int i = 1, j = 2, k = 3;
double x = 4.5, y = 5.5;
```

Pour chacune des instanciations ci-dessous, dire si celle-ci est correcte ou non et si elle l'est, indiquer quel est le type et la valeur du résultat livré en retour.

1) `f(i, x, j)`
2) `f<>(x, i, y)`
3) `f(i, j, k)`
4) `f(i, j, x)`
5) `f((double)i, j, x)`
6) `f<int, double>(i, j, x)`
7) `f<int>(i, x, x)`
8) `f<double>(i, x, x)`

Exercice 8.2 Occurrences d'une valeur dans un tableau

Ecrire une fonction générique C++ *nbOcc* qui prend en paramètre un tableau classique à 1 dim d'éléments d'un type quelconque et qui livre en retour combien de fois un certain élément figure dans le tableau.

Appliquer la fonction pour rechercher :

1) la valeur 0 dans le tableau de *int* : [0, 1, 0]
2) le temps[1] 12:00 dans le tableau de temps : [07:45, 09:20, 12:00, 21:30]
   [1] à implémenter sous forme d'une classe
3) la chaîne "Paul" dans le tableau de *string* : [Paul, Jacques, Paul, Jean, Paul]
4) la chaîne "Paul" dans le tableau de *const char** : [Paul, Jacques, Paul, Jean, Paul]

NB Le point 4) est une question difficile

Exercice 8.3 Surcharge et spécialisation (1)

Soient les définitions de fonctions suivantes :

```
template <typename T, typename U> void f(T, U) {...}   // fonction I
template <typename T, typename U> void f(T*, U) {...}  // fonction II
template <typename T> void f(T, T) {...}               // fonction III
template <typename T> void f(T, int) {...}             // fonction IV
void f(int, int) {...}                                 // fonction V
void f(int*, float) {...}                              // fonction VI
```

et soient les déclarations de variables suivantes :

```
char c = 'A';
int i = 1, j = 2;
float x = 3.f, y = 4.f;
double z = 5.0;
```

Pour chacun des appels ci-dessous, dire si celui-ci est correct ou non et s'il l'est, indiquer quelle fonction est appelée.

1) `f(i, j);`
2) `f(c, i);`
3) `f(x, y);`
4) `f(i, z);`
5) `f(&i, j);`
6) `f(&i, x);`
7) `f(&i, z);`

Exercice 8.4 Surcharge et spécialisation (2)

1) Soient les définitions de fonctions suivantes :

```
template <typename T, typename U> void f(T, U, int); // fonction 1
template <typename T> void f(T, T, short);           // fonction 2
template <typename T> void f(T, int, int);           // fonction 3
void f(int, int, int);                               // fonction 4
```

et soient les déclarations de variables suivantes :

```
char c = 'A';
short s = 1;
int i = 2, j = 3;
float x = 4.f;
```

Pour chacun des appels ci-dessous, dire si celui-ci est correct ou non et s'il l'est, indiquer quelle fonction est appelée.

a) `f(i, j, c);`
b) `f(i, j, s);`
c) `f(i, j, x);`

2) Même question que la question 1) mais on suppose ne disposer que des fonctions 2, 3 et 4
3) Même question que la question 1) mais on suppose ne disposer que des fonctions 1, 2 et 3
4) Même question que la question 1) mais on suppose ne disposer que des fonctions 1 et 2

Exercice 8.5 Instanciations d'une classe générique

Soient les deux classes génériques (ou *patrons de classes*) suivantes :

```
template <typename T = int> class A {…};
template <typename T, typename U, int n> class B {…};
```

Pour chacune des instanciations ci-dessous, dire si celle-ci est correcte ou non.

1) `B<int, double, 1> b;`

2)
```
const int N = 1;
B<int, int, N> b;
```

3)
```
int n = 1;
B<int, double, n> b;
```

4) `B<int, double, 'A'> b;`

5) `B<int, double, 'A' + 'B'> b;`

6) `B<int, double, 1.0> b;`

7) `B<int, int*, 1> b;`

8) `B<A<>, double, 1> b;`

9) `B<double, A, 1> b;`

10) `B<A<double>, double, true> b;`

Exercice 8.6 Classe générique Array (1)

Compléter les trois parties notées ***< à compléter >*** du programme ci-dessous, de telle sorte que celui-ci affiche, à l'exécution :

```
[1, 1, 1]
[1.5, 1.5, 1.5, 1.5]
```

```
< à compléter 1 >

template <typename T, size_t n> class Array {
public:
   Array(const T& valeur);
   < à compléter 2 >
private:
   T tab[n];
};

< à compléter 3 >

int main() {
   Array<int, 3> a1(1);
   a1.afficher();
   cout << endl;

   Array<double, 4> a2(1.5);
   a2.afficher();
   cout << endl;

   return EXIT_SUCCESS;
}
```

Exercice 8.7 Classe générique Array (2)

Compléter les trois parties notées ***< à compléter >*** du programme ci-dessous, de telle sorte que celui-ci affiche, à l'exécution :

```
[1, 1, 1]
[1.5, 1.5, 1.5, 1.5]
```

```
< à compléter 1 >

template <typename T, size_t n> class Array {
   < à compléter 2 >
public:
   Array(const T& valeur);
private:
   T tab[n];
};

< à compléter 3 >

int main() {
   Array<int, 3> a1(1);
   cout << a1 << endl;

   Array<double, 4> a2(1.5);
   cout << a2 << endl;

   return EXIT_SUCCESS;
}
```

Exercice 8.8 Classe générique Array (3)

Compléter les trois parties notées **< *à compléter* >** du programme ci-dessous, de telle sorte que celui-ci affiche, à l'exécution :

```
[1, 1, 1]
[1.5, 1.5, 1.5, 1.5]
[true, true, true, true, true]
```

```
< à compléter 1 >

template <typename T, size_t n> class Array {
   < à compléter 2 >
public:
   Array(const T& valeur);
private:
   T tab[n];
};

< à compléter 3 >

int main() {
   Array<int, 3> a1(1);
   cout << a1 << endl;

   Array<double, 4> a2(1.5);
   cout << a2 << endl;

   Array<bool, 5> a3(true);
   cout << a3 << endl;

   return EXIT_SUCCESS;
}
```

Exercice 8.9 Spécialisations

1) Que va afficher, à l'exécution, le programme ci-dessous ?

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

template <typename T> class C {
public:
   C(T t) : t(t) {};
   void afficher() const;
private:
   T t;
};

template <typename T>
void C<T>::afficher() const {
   cout << t;
}

int main() {

   int n = 1;

   C<int> c1(n);
   c1.afficher();
   cout << endl;

   C<int*> c2(&n);
   c2.afficher();
   cout << endl;

   C<const char*> c3("Hello");
   c3.afficher();
   cout << endl;

   return EXIT_SUCCESS;
}
```

2) Comment faut-il modifier le code ci-dessus (sans toucher à *main*) pour que celui-ci affiche à l'exécution :

```
1
1
Hello
```

Exercice 8.10 Collection générique

Compléter la partie notée ***< à compléter >*** du programme ci-dessous, de telle sorte que celui-ci affiche, à l'exécution :

```
integers contient 3 elements : [1, 2, 3]
integers contient 0 element : []
strings contient 4 elements : [un, deux, trois, quatre]
strings contient 0 element : []
```

```
< à compléter >

int main() {

   MaCollection<int, vector> integers;
   size_t nb_integers;

   integers.add(1);
   integers.add(2);
   integers.add(3);

   nb_integers = integers.size();
   cout << "integers contient " << nb_integers
        << " element" << (nb_integers > 1 ? "s" : "") << " : ";
   afficher(integers);
   integers.clear();
   nb_integers = integers.size();
   cout << "integers contient " << nb_integers
        << " element" << (nb_integers > 1 ? "s" : "") << " : ";
   afficher(integers);


   MaCollection<string, list> strings;
   size_t nb_strings;

   strings.add("un");
   strings.add("deux");
   strings.add("trois");
   strings.add("quatre");

   nb_strings = strings.size();
   cout << "strings contient " << nb_strings
        << " element" << (nb_strings > 1 ? "s" : "") << " : ";
   afficher(strings);
   strings.clear();
   nb_strings = strings.size();
   cout << "strings contient " << nb_strings
        << " element" << (nb_strings > 1 ? "s" : "") << " : ";
   afficher(strings);

   return EXIT_SUCCESS;
}
```

Chapitre 9 : Exceptions

Exercice 9.1 try / catch (1)

Que va produire, à l'exécution, le programme ci-dessous ?

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

int main() {

   try {
      throw 1;
   } catch (int e) {
      cout << "Dans catch (int) : valeur recue = " << e << endl;
      throw 2.0;
   } catch (double e) {
      cout << "Dans catch (double e) : valeur recue = " << e << endl;
   } catch (...) {
      cout << "Dans catch (...)" << endl;
   }
   cout << "Fin du programme" << endl;

   return EXIT_SUCCESS;
}
```

Exercice 9.2 try / catch (2)

Que va produire, à l'exécution, le programme ci-dessous ?

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

int main() {
   try {
      try {
         throw 1;
      } catch (int e) {
         cout << "Dans catch (int) : valeur recue = " << e << endl;
         throw 2.0;
      } catch (...) {
         cout << "Dans catch (...)" << endl;
      }
   } catch (double e) {
      cout << "Dans catch (double e) : valeur recue = " << e << endl;
   }
   cout << "Fin du programme" << endl;

   return EXIT_SUCCESS;
}
```

Exercice 9.3 try / catch (3)

Que va produire, à l'exécution, le programme ci-dessous ?

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

void f() {
   throw 'A';
}

int main() {
   try {
      try {
         f();
      } catch (int) {
         cout << "Dans catch (int) 1" << endl;
         throw;
      } catch (...) {
         cout << "Dans catch (...) 1" << endl;
         throw 65;
      }
   } catch (int&) {
      cout << "Dans catch (int&)" << endl;
   } catch (int) {
      cout << "Dans catch (int) 2" << endl;
   } catch (const int) {
      cout << "Dans catch (const int)" << endl;
   } catch (...) {
      cout << "Dans catch (...) 2" << endl;
   }

   cout << "Fin du programme" << endl;

   return EXIT_SUCCESS;
}
```

Exercice 9.4 try / catch (4)

Que va produire, à l'exécution, le programme ci-dessous ?

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

void f() {
   throw out_of_range("out of range");
}

int main() {
   try {
      try {
         f();
      } catch (runtime_error& e) {
         cout << e.what() << endl;
         throw;
      } catch (exception e) {
         cout << e.what() << endl;
         throw;
      }
   } catch (logic_error& e) {
      cout << e.what() << endl;
   } catch (exception& e) {
      cout << e.what() << endl;
   }

   cout << "Fin du programme" << endl;

   return EXIT_SUCCESS;
}
```

Exercice 9.5 Construction d'un objet membre (1)

Sans utiliser le concept de *function-try block*, compléter les deux constructeurs des classes *Identite* et *Personne* de telle sorte que, à l'exécution, le programme ci-dessous affiche :

```
Debut du test 1.
Tentative de construction d'un objet du type Personne.
Parametres fournis: "John Fitzgerald", "Kennedy"
Dans Identite::Identite(): John Fitzgerald Kennedy.
Dans Personne::Personne(): John Fitzgerald Kennedy.
Fin du test 1.

Debut du test 2.
Tentative de construction d'un objet du type Personne.
Parametres fournis: "", "Marley"
Exception survenue dans Identite::Identite():
prenom ne peut pas etre une chaine vide.
Exception survenue dans Personne::Personne().
Exception survenue dans main().
```

IMPORTANT

- Hormis les deux constructeurs à compléter, aucune ligne de code ne doit être ajoutée, modifiée ou supprimée du programme proposé ci-dessous

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

class Identite {
public:
   Identite() = default;

   Identite(const string& prenom, const string& nom)
   <à compléter 1>

   string toString() const {
      return prenom + " " + nom;
   }

private:
   string prenom;
   string nom;
};

class Personne {
public:
   Personne(const string& prenom, const string& nom)
   <à compléter 2>

private:
   Identite identite;
};
```

```
int main() {
   try {
      {
         cout << "Debut du test 1." << endl
              << "Tentative de construction d'un objet du type Personne." << endl
              << "Parametres fournis: \"John Fitzgerald\", \"Kennedy\"" << endl;
         Personne personne1("John Fitzgerald", "Kennedy");
         cout << "Fin du test 1." << endl;
      }
      {
         cout << endl
              << "Debut du test 2." << endl
              << "Tentative de construction d'un objet du type Personne." << endl
              << "Parametres fournis: \"\", \"Marley\"" << endl;
         Personne personne2("", "Marley");
         cout << "Fin du test 2." << endl;
      }
   } catch (const invalid_argument&) {
      cout << "Exception survenue dans main()." << endl;
   }

   return EXIT_SUCCESS;
}
```

Exercice 9.6 Construction d'un objet membre (2)

Même énoncé que celui de l'exercice 9.5, sauf qu'ici il est demandé d'utiliser le concept de *function-try block*.

Exercice 9.7 Construction d'un objet membre (3)

Même énoncé que celui de l'exercice 9.5, sauf qu'ici on suppose les champs *prénom* et *nom* de *Identite* ainsi que le champ *identite* de *Personne* comme étant déclarés constants.

Exercice 9.8 Construction d'un objet membre (4)

Même énoncé que celui de l'exercice 9.6, sauf qu'ici on suppose les champs *prénom* et *nom* de *Identite* ainsi que le champ *identite* de *Personne* comme étant déclarés constants.

Exercice 9.9 Somme des n premiers entiers naturels (1)

Ecrire un programme C++ qui met à disposition une fonction *sommeNPremiersEntiers* permettant de sommer les *n* premiers entiers ≥ 0 et dont le prototype est

```
int sommeNPremiersEntiers(int n);
```

Faire en sorte d'implémenter *sommeNPremiersEntiers* en tenant compte de toutes les situations problématiques potentielles.

IMPORTANT
L'exercice doit être résolu en utilisant exclusivement une(des) exception(s) prédéfinie(s).

Exercice 9.10 Somme des n premiers entiers naturels (2)

Même énoncé que celui de l'exercice 9.9, sauf qu'ici il est demandé de résoudre l'exercice en implémentant votre(vos) propre(s) classe(s)-exceptions.

Exercice 9.11 Somme des n premiers entiers naturels (3)

Même énoncé que celui de l'exercice 9.9, sauf qu'ici on suppose que le prototype de la fonction *sommeNPremiersEntiers* est :

```
unsigned sommeNPremiersEntiers(unsigned n);
```

Expliquer en quoi les solutions des exercices 9.9 et 9.11 diffèrent.

Exercice 9.12 Insertion d'une valeur dans un tableau

Ecrire un programme C++ qui met à disposition une fonction *void* permettant d'insérer (en préservant l'ordre des éléments) une valeur à une position *pos* donnée dans un tableau *tab* classique à 1 dimension de *int*.

Si lors de l'appel à la fonction on ne donne pas de valeur pour le paramètre *pos*, l'insertion se fait au début de *tab*.

Faire en sorte que la fonction signale (en levant une exception prédéfinie) tout problème susceptible de se produire.

Exercice 9.13 Exception avec info

Ecrire un programme C++ qui :

1) met à disposition une fonction sans paramètre et sans valeur de retour dont la seule tâche est de lever une exception comprenant 2 informations : un no d'erreur (de type *int*) et un message d'erreur (de type *string*)
2) appelle cette fonction
3) récupère l'exception levée par la fonction dans un traite-exceptions.
   Supposer que la seule tâche du traite-exceptions consiste à afficher les informations (no d'erreur et message) "véhiculées" par l'exception interceptée.

Exercice 9.14 Terminaison de programme (1)

Que va afficher le programme ci-dessous, lorsque l'utilisateur saisit :

1) la valeur 1 ?
2) la valeur 0 ?
3) la valeur 2 ?

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

void onExit() {
   cout << "Appel de exit\n";
   system("PAUSE");
}

int main() {

   atexit(onExit);

   int n;

   cout << "Donnez un entier : ";
   cin >> n;

   try {
      cout << "Debut du premier bloc try\n";
      if (n) throw n;
      cout << "Fin du premier bloc try\n";
   } catch (int n) {
      if (n == 1)
         cout << "catch 1 : n = " << n << endl;
      else
         exit(EXIT_FAILURE);
   }

   cout << "... suite du programme\n";

   try {
      cout << "Debut du second bloc try\n";
      throw n;
      cout << "Fin du second bloc try\n";
   } catch (int n) {
      cout << "catch 2 : n = " << n << endl;
   }

   cout << "Fin du programme\n";

   return EXIT_SUCCESS;
}
```

Exercice 9.15 Terminaison de programme (2)

Que va afficher le programme ci-dessous, lorsque l'utilisateur saisit :

1) la valeur 1 ?
2) la valeur 2 ?
3) la valeur 3 ?
4) la valeur 4 ?

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

void onExit() {
   cout << "Appel de exit\n";
   system("PAUSE");
}

void onTerminate() {
   cout << "Appel de terminate\n";
   // exit(EXIT_FAILURE); // pour éviter que le programme ne "plante"
}

int main() {

   atexit(onExit);
   set_terminate(onTerminate);

   int n;
   float x;
   double y;

   cout << "Donnez un entier : ";
   cin >> n;

   try {
      switch (n) {
         case 1: throw n;
         case 2: x = n; throw x;
         case 3: y = n; throw y;
      }
   } catch (int n) {
      cout << "catch(int n) : n = " << n << endl;
   } catch (float x) {
      cout << "catch(float x) : x = " << x << endl;
      exit(EXIT_FAILURE);
   }

   cout << "Fin du programme\n";

   return EXIT_SUCCESS;
}
```

Exercice 9.16 Terminaison de programme (3)

1) Que va afficher ce programme ?

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

void f();

int main() {

   try {
      f();
   } catch (int n) {
      cout << "Exception int dans main : " << n << endl;
   } catch (...) {
      cout << "Exception autre que int dans main" << endl;
   }

   cout << "Fin main\n";
   return EXIT_SUCCESS;
}

void f() {
   try {
      int n = 1;
      throw n;
   } catch (int n) {
      cout << "Exception int dans f : " << n << endl;
      throw;
   }
}
```

2) Même question si l'on remplace *f* par :

```
void f() {
   try {
      double x = 1.;
      throw x;
   } catch (int n) {
      cout << "Exception int dans f : " << n << endl;
      throw;
   }
}
```