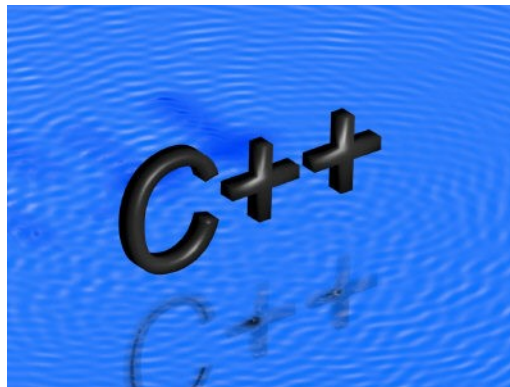


Département TIC

INF2

Recueil d'exercices C++



René Rentsch

Table des matières

*Les exercices avec * sont les exercices pour lesquels un corrigé est distribué aux étudiants*

Chapitre 7 : Classes	1
Exercice 7.1 Pays (1)	1
Exercice 7.2 Pays (2)	1
Exercice 7.3 Robot	2
Exercice 7.4* Point (1)	2
Exercice 7.5 Point (2)	3
Exercice 7.6 Point (3)	3
Exercice 7.7 Point (4)	3
Exercice 7.8* Membres statiques	4
Exercice 7.9 Voitures	5
Exercice 7.10 Messages	6
Exercice 7.11 Mailbox	7
Exercice 7.12 Personnes	8
Exercice 7.13 Constructeur de (re)copie et opérateur d'affectation	10
Exercice 7.14 Surcharges d'opérateurs	11
Chapitre 8 : Généricité	12
Exercice 8.1 Instanciations d'une fonction générique	12
Exercice 8.2* Occurrences d'une valeur dans un tableau	12
Exercice 8.3 Surcharge et spécialisation (1)	13
Exercice 8.4 Surcharge et spécialisation (2)	14
Exercice 8.5 Instanciations d'une classe générique	15
Exercice 8.6* Classe générique Array (1)	16
Exercice 8.7 Classe générique Array (2)	17
Exercice 8.8 Spécialisations	18
Exercice 8.9 Collection générique	19
Chapitre 9 : Exceptions	20
Exercice 9.1 try / catch (1)	20
Exercice 9.2 try / catch (2)	20
Exercice 9.3 try / catch (3)	21
Exercice 9.4 try / catch (4)	22

Exercice 9.5*	Construction d'un objet membre (1).....	23
Exercice 9.6	Construction d'un objet membre (2).....	24
Exercice 9.7	Construction d'un objet membre (3).....	24
Exercice 9.8	Construction d'un objet membre (4).....	24
Exercice 9.9	Somme des n premiers entiers naturels (1)	25
Exercice 9.10	Somme des n premiers entiers naturels (2)	25
Exercice 9.11	Somme des n premiers entiers naturels (3)	25
Exercice 9.12*	Insertion d'une valeur dans un tableau	25
Exercice 9.13	Exception avec info	26
Exercice 9.14	Terminaison de programme (1).....	27
Exercice 9.15	Terminaison de programme (2).....	28
Exercice 9.16	Terminaison de programme (3).....	29
Chapitre 10 : Introduction au C.....		31
Exercice 10.1	Substitutions	31
Exercice 10.2	Macros (1)	31
Exercice 10.3	Macros (2)	32
Exercice 10.4	Opérateurs de manipulation de bits (1)	33
Exercice 10.5*	Récupérer la valeur du nième bit.....	33
Exercice 10.6	Fixer la valeur du nième bit.....	33
Exercice 10.7	Position du bit à 1 de plus faible poids.....	34
Exercice 10.8	Représentation binaire d'un entier	34
Exercice 10.9*	printf (1)	34
Exercice 10.10	printf (2)	35
Exercice 10.11	printf (3)	35
Exercice 10.12*	scanf.....	36
Chapitre 11 : Pointeurs.....		37
Exercice 11.1*	Les bases... ..	37
Exercice 11.2	Paramètres d'entrée-sortie	37
Exercice 11.3	Fonction avec valeur de retour de type pointeur	37
Exercice 11.4*	Interprétation de déclarateurs avec pointeurs	38
Exercice 11.5*	Ecriture de déclarateurs avec pointeurs.....	38
Exercice 11.6	Affectations entre pointeurs	39
Exercice 11.7	Arithmétique des pointeurs (1).....	40
Exercice 11.8	Arithmétique des pointeurs (2).....	40

Exercice 11.9	Arithmétique des pointeurs (3).....	41
Exercice 11.10	Arithmétique des pointeurs (4).....	42
Exercice 11.11	Formalisme pointeur et tableaux 1D	42
Exercice 11.12	Inversion d'un tableau 1D (1).....	42
Exercice 11.13	Méli-mélo	43
Exercice 11.14	Pointeurs, tableaux et fonctions	44
Exercice 11.15*	Utilisation de calloc	44
Exercice 11.16	Inversion d'un tableau 1D (2).....	45
Exercice 11.17	Adresses du min et du max d'un tableau 1D	45
Exercice 11.18	Manipulation de la mémoire (1).....	45
Exercice 11.19	Manipulation de la mémoire (2).....	46
Exercice 11.20	Manipulation de la mémoire (3).....	46
Exercice 11.21	Initialisation d'une matrice	47
Exercice 11.22	Somme des valeurs d'une matrice	47
Exercice 11.23	Diagonale d'une matrice carrée	47
Exercice 11.24	Calcul d'une intégrale	48
Chapitre 12 : Types composés		49
Exercice 12.1	Correction d'erreurs	49
Exercice 12.2*	Affichage d'une personne (1)	49
Exercice 12.3	Affichage de plusieurs personnes.....	50
Exercice 12.4	Lendemain d'une date	50
Exercice 12.5*	Taille d'une structure	51
Exercice 12.6	Affichage d'une personne (2).....	52
Exercice 12.7	Bateaux.....	53
Exercice 12.8	Affichage des jours de la semaine (1)	54
Exercice 12.9	Affichage des jours de la semaine (2)	54
Exercice 12.10	Pile dynamique.....	55
Chapitre 13 : Chaînes de caractères		56
Exercice 13.1*	Implémentation de strlen.....	56
Exercice 13.2	Implémentation de strcpy	56
Exercice 13.3	Implémentation de strncpy	56
Exercice 13.4	Implémentation de strcat	57
Exercice 13.5	Implémentation de strncat	57
Exercice 13.6*	Implémentation de strcmp	57

Exercice 13.7	Implémentation de strechr.....	58
Exercice 13.8	Prénom et nom	58
Exercice 13.9	Inversion d'une chaîne	58
Chapitre 14 : Fichiers	59
Exercice 14.1*	Lecture intégrale d'un fichier texte.....	59
Exercice 14.2	Ecriture d'un fichier binaire.....	59
Exercice 14.3	Lecture intégrale d'un fichier binaire	59
Exercice 14.4	Recherche séquentielle dans un fichier binaire	60
Exercice 14.5	Recherche par accès direct dans un fichier binaire	60
Exercice 14.6*	Compteurs d'octets	61
Exercice 14.7	Lecture d'un fichier texte sans utiliser de boucle	61

Chapitre 7 : Classes

Exercice 7.1 Pays (1)

- 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 sans faire appel à la compilation séparée
- Des données réelles peuvent être trouvées sous
<http://www.statistiques-mondiales.com/europe.htm>

Exercice 7.2 Pays (2)

Même énoncé que l'exercice 7.1 mais il est demandé ici d'exploiter la compilation séparée.

Exercice 7.3 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 7.4* 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 7.5 Point (2)

Implémenter une classe *Point*, analogue à celle de l'exercice 7.4, 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 7.4 pour qu'il fonctionne avec cette nouvelle version de la classe *Point*.

Exercice 7.6 Point (3)

Ajouter à la classe *Point* de l'exercice 7.5 (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 7.7 Point (4)

Ajouter à la classe *Point* de l'exercice 7.5 (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 7.8* 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
- 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.

Important Cette fonction membre doit utiliser la fonction membre *toString*

- 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 prédéfinie *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 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.
 - L'amitié est réciproque (si on ajoute un ami *A* à *B*, alors *B* doit se voir automatiquement ajouté comme ami de *A*)
 - 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 chaîne 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.

Exercice 7.14 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

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 temps¹ 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 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 8.9 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 personnel("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 9.7 et 9.9 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";

    system("PAUSE");
    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;
    }
}
```

3) Même question que la question 2) mais en remplaçant en plus dans le code toutes les occurrences de *void f()* par *void f() throw (int)*

4) Que va afficher le code suivant ?

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

void f() throw (int, char);

void onUnexpected() {
    throw 'q';
}

int main() {

    set_unexpected(onUnexpected);

    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";

    system("PAUSE");
    return EXIT_SUCCESS;
}

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

Chapitre 10 : Introduction au C

Exercice 10.1 Substitutions

En supposant les `<?>` du code ci-dessous remplacés par le code de format adéquat, indiquer ce que va afficher le programme suivant ?

```
#include <stdio.h>
#include <stdlib.h>

#define A 1
#define B 2
#define AB 3
#define F(X,Y) X##Y

int main(void) {

    int n = AB;

    printf("%<?>\n", n);
    printf("%<?>\n", "AB");
    printf("%<?>\n", F(A, B));

    return EXIT_SUCCESS;
}
```

Exercice 10.2 Macros (1)

Le programme suivant contient une ou plusieurs erreurs. Corrigez-le de telle sorte qu'il affiche à l'exécution :

25 -1

```
#include <stdio.h>
#include <stdlib.h>

#define A = 2;
#define B = A + 1;
#define PLUS (X,Y) X+Y;
#define MOINS(X,Y) X-Y;

int main(void) {

    printf("%d %d\n", 5*PLUS(A,B), MOINS(A+1,B+1));

    return EXIT_SUCCESS;
}
```

Exercice 10.3 Macros (2)

Soit le programme suivant (ABS = macro "valeur absolue") :

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#define ABS(X) (((X) > 0) ? (X) : (-X))

int main(void) {

    int n;

    printf("1. %d\n", ABS(2));
    printf("2. %d\n", ABS(-2));
    printf("3. %d\n", ABS('A'));

    n = -2;
    printf("4. %d\n", ABS(n+1));

    n = -2;
    printf("5. %d", ABS(n++)); printf(" %d\n", n);

    n = -2;
    printf("6. %d", ABS(++n)); printf(" %d\n", n);

    n = -2;
    printf("7. %d", abs(++n)); printf(" %d\n", n);

    return EXIT_SUCCESS;
}
```

- 1) Que va afficher ce programme ?
- 2) Expliquez en quoi le programme ci-dessus est problématique.
- 3) Est-il possible de récrire (sans utiliser la fonction `abs` de `stdlib`) la macro `ABS` de manière à éviter le(s) problème(s) du point 2) ?

Exercice 10.4 Opérateurs de manipulation de bits (1)

Que vaut chacune des expressions suivantes ?

- 1) $22 \mid 11$
- 2) $0 \wedge 19$
- 3) $2 \ll 3$
- 4) $30 \& 14$
- 5) $8 \gg 2$
- 6) $4 \& 29$
- 7) $9 \ll 4$
- 8) $31 \wedge 27$
- 9) $23 \mid 3$
- 10) $-1 \gg 1$
- 11) $-5 \gg 1$
- 12) $3 \& \sim 2$
- 13) $6 \mid 5 \& 4$

Exercice 10.5* Récupérer la valeur du nième bit

En utilisant exclusivement des opérateurs de manipulation de bits, implémenter la fonction dont le prototype et la sémantique sont donnés ci-dessous.

```
unsigned short getBit(unsigned short pos, int n);
```

Sémantique

Retourne la valeur du bit en position *pos* dans la représentation binaire du nombre *n*

NB *pos* = 0 correspond à la position du bit de poids le plus faible

Exemples : $\text{getBit}(0, 5) = 1$; $\text{getBit}(1, 5) = 0$; $\text{getBit}(2, 5) = 1$

Exercice 10.6 Fixer la valeur du nième bit

En utilisant exclusivement des opérateurs de manipulation de bits, implémenter la fonction dont le prototype et la sémantique sont donnés ci-dessous.

```
int* setBit(unsigned short pos, unsigned short bitValue, int* n);
```

Sémantique

Met le bit en position *pos* dans la représentation binaire du nombre *n* à la valeur *bitValue*

NB *pos* = 0 correspond à la position du bit de poids le plus faible

Exercice 10.7 Position du bit à 1 de plus faible poids

En utilisant exclusivement des opérateurs de manipulation de bits, implémenter la fonction dont le prototype et la sémantique sont donnés ci-dessous.

```
short lowestOrderSetBit(int n);
```

Sémantique

Retourne la position du bit à 1 de plus faible poids. Retourne -1 si aucun bit n'est à 1.

NB $pos = 0$ correspond à la position du bit de poids le plus faible

Exercice 10.8 Représentation binaire d'un entier

En exploitant au maximum les opérateurs de manipulation de bits, implémenter la fonction dont le prototype et la sémantique sont donnés ci-dessous.

```
void decimalToBinary(int32_t n, int8_t binary[]);
```

Sémantique

Convertit en binaire le nombre entier décimal n et place le résultat dans le tableau *binary*.

Exercice 10.9* printf (1)

Que va afficher le programme ci-dessous ?

(Indiquez par *b* la présence d'un espace blanc)

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {

    int n = 127;
    double x = 12.3456;

    printf("R1 : |%-4d|\n", n);
    printf("R2 : |%04d|\n", n);
    printf("R3 : |%x|\n", n);
    printf("R4 : |%#o|\n", n);
    printf("R5 : |%f|\n", x);
    printf("R6 : |%5.2f|\n", x);
    printf("R7 : |%.2e|\n", x);
    printf("R8 : |%g|\n", x);

    return EXIT_SUCCESS;
}
```


Exercice 10.10 printf (2)

Soit le squelette de code suivant :

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {

    int n = 255;
    double x = 12.345;

    < à compléter>

    return EXIT_SUCCESS;
}
```

Compléter le code ci-dessus de sorte qu'à l'exécution il reproduise à l'identique l'output suivant :

```
0377
FF
+###255
1.235e+001
12.345
12.3450
```

Exercice 10.11 printf (3)

Compléter le code ci-dessous de telle sorte que celui-ci affiche à l'exécution :

```
i = 1
j = 4294967295
```

Important Le code, une fois complété, doit compiler sans warnings.

```
#include <stdio.h>
#include <stdlib.h>
<à compléter>

int main(void) {

    size_t i = 1;
    uint32_t j = UINT32_MAX;

    <à compléter>

    return EXIT_SUCCESS;
}
```

Exercice 10.12* scanf

Le code ci-dessous n'est pas correct. Le récrire complètement de sorte à corriger toutes les erreurs / maladresses commises.

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

int main() {

    int n, char c;

    printf("Donnez un nombre entier et un caractere : ");
    scanf("%d%c", n, c);
    printf("n = %d, c = %c\n", n, c);

    return EXIT_SUCCESS;
}
```

Chapitre 11 : Pointeurs

Exercice 11.1* Les bases...

Ecrire un programme C qui :

- 1) Déclare une variable *n* de type *int* et l'initialise à la valeur 1
- 2) Déclare un pointeur *ptr* pointant sur *n*
- 3) Affiche à l'écran la valeur de l'objet pointé par *ptr*
- 4) Affiche à l'écran l'adresse contenue dans *ptr*
- 5) Affiche l'adresse de *ptr*

Exercice 11.2 Paramètres d'entrée-sortie

Ecrire une fonction C sans valeur de retour (*void*) qui prend comme paramètre d'entrée un nombre réel *x* (de type *double*) et qui renvoie, par paramètres d'entrée-sortie, le carré et le cube de *x*.

Exercice 11.3 Fonction avec valeur de retour de type pointeur

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

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int* carres(const int tab[]);

int main(void) {
    int tab[] = {1, 2, 3};
    const size_t N = sizeof(tab) / sizeof(int);
    int* ptr = carres(tab);
    for (size_t i = 0; i < N; ++i)
        printf("%d ", ptr[i]);
    printf("\n");
    return EXIT_SUCCESS;
}

int* carres(const int tab[]) {
    const size_t N = sizeof(tab) / sizeof(int);
    int copie[N];
    memcpy(copie, tab, N * sizeof(int));
    for (size_t i = 0; i < N; ++i)
        copie[i] *= copie[i];
    return copie;
}
```

Exercice 11.4* Interprétation de déclarateurs avec pointeurs

Traduire en français les déclarations C suivantes :

(Exemple : pour `int* ptr`, il faut répondre "*ptr est un pointeur sur int*")

- 1) `char** a[10];`
- 2) `double* (*b)(void);`
- 3) `double (*c)(double*);`
- 4) `int* d[10];`
- 5) `int (*e)[10];`
- 6) `int (*(*f[5])(void))[10];`
- 7) `double (**g)[5];`

Exercice 11.5* Ecriture de déclarateurs avec pointeurs

Ecrire les déclarations C correspondant aux énoncés suivants :

- 1) t est un tableau de 10 pointeurs pointant chacun sur un *int* constant
- 2) t est un tableau de 10 pointeurs constants pointant chacun sur un *int*
- 3) p est un pointeur sur une fonction prenant en paramètre un pointeur sur une fonction (prenant en paramètre un *double* et livrant un *double*) et renvoyant un pointeur sur *int*
- 4) p est un pointeur constant sur un tableau de 10 pointeurs sur *double*
- 5) t est un tableau de 10 pointeurs pointant chacun sur une fonction prenant un *double* comme paramètre et renvoyant un pointeur sur *char*
- 6) f est une fonction prenant en paramètre un pointeur constant sur *char* et renvoyant un pointeur constant contenant l'adresse d'un pointeur sur *char*
- 7) f est une fonction sans paramètre renvoyant un pointeur sur un tableau de 10 pointeurs constants sur *char*

Exercice 11.6 *Affectations entre pointeurs*

Soient les déclarations suivantes :

```
int i = 1, j = 2;  
int* pi1 = &i;  
int* pi2 = &j;  
double x = 3.0;  
double* pd = &x;  
void* pv;
```

Quel sera dans un programme l'effet des instructions suivantes ?

- 1) pi1 = pi2;
- 2) pd = pi1;
- 3) pi1 = pd;
- 4) pv = pi1;
- 5) pv = &i;
- 6) pv = pi1;
 pi2 = pv;
- 7) if (pi1 = pi2)...

Exercice 11.7 Arithmétique des pointeurs (1)

Que va afficher le programme *main* ci-dessous ?

(Conseil : Aidez-vous d'un petit dessin)

```
#include <inttypes.h>
#include <stdlib.h>
#include <stdio.h>

int main(void) {

    int a[] = {10, 20, 30, 40, 50};
    int* p[] = {a, a+1, a+2, a+3, a+4};
    int** pp = &p[2];
    int*** ppp = &pp;

    printf("1) %d\n", *a);
    printf("2) %d\n", **p);
    printf("3) %d\n", **pp++);
    printf("4) %d\n", ***ppp);
    pp = p;
    printf("5) %d\n", **++pp);
    pp = p;
    printf("6) %d\n", ++**pp);
    pp = p;
    printf("7) %d\n", *pp[1]);
    pp = p;
    printf("8) %" PRIuPTR "\n", (ptrdiff_t) (*(p+1) - *pp));

    return EXIT_SUCCESS;
}
```

Exercice 11.8 Arithmétique des pointeurs (2)

Que va afficher le programme *main* ci-dessous ?

(Conseil : Aidez-vous d'un petit dessin)

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {

    int t[6], i, *p;

    for (p = &t[5]; p > &t[-1]; p--)
        *p = (int) (p - t);

    for (i = 0; i < 6; i++)
        printf("%d ", t[i]);
    printf("\n");

    for (i = 0, ++p; i < 6; i++)
        printf("%d ", *p++/2);
    printf("\n");

    return EXIT_SUCCESS;
}
```

Exercice 11.9 Arithmétique des pointeurs (3)

Soient les déclarations suivantes :

```
const char* c[] = {"elle", "mangera", "une", "petite", "tomme"};  
const char** d[] = {c+1, c+2, c+3, c+4, c};  
const char*** e = &d[3];
```

Quelles valeurs fournissent les expressions suivantes ?

(Conseil : Aidez-vous d'un petit dessin)

- a) `c[3][0]`
- b) `(**d)[5]`
- c) `(**e)[*d-c]`
- d) `(d[3]-3)[0][3]`
- e) `**d + 5`
- f) `*d[3] + 2`
- g) `*(*e[-3] + 5)`
- h) `**c`
- i) `e[0][0][e-d]+1`
- j) `0[c][0] - 'd' + 'B'`

Remarques

- Pour les chaînes de caractères (`char *`), donner le résultat entre guillemets.
Ex : "ABC".
- Pour les caractères (`char`), donner la représentation littérale. Ex : 'a'.

Exercice 11.10 Arithmétique des pointeurs (4)

Que va afficher le programme *main* ci-dessous ?

(Conseil : Aidez-vous d'un petit dessin)

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {

    const char* c[] = {"comprendre", "les", "pointeurs", "c'est", "difficile"};
    const char** cp[] = {c, c + 2, c + 4, c + 1, c + 3};
    const char*** cpp = cp;
    int i;

    for (i = 0; i < 3; i++)
        printf("%c", *(*cpp + i));
    printf("%c", *(*cp[0] + 2));
    printf("%s ", *cpp[2] + 8);
    printf("%s ", *++++cpp);
    for (i = 1; i < 4; i++)
        printf("%c", *(cpp[-1][i % 3] + 2));
    printf("%c", **--*cpp);
    printf("%s\n", *+++cpp + 7);

    return EXIT_SUCCESS;
}
```

Exercice 11.11 Formalisme pointeur et tableaux 1D

Ecrire d'au moins 6 manières différentes, mais toujours en utilisant le formalisme pointeur, une fonction C *initialiser*, sans valeur de retour (*void*), permettant d'initialiser à une valeur donnée (de type *int*), tous les éléments d'un tableau à 1 dimension (1D) de taille quelconque et composé d'entiers (de type *int*).

Exercice 11.12 Inversion d'un tableau 1D (1)

Implémenter la fonction C dont le prototype et la sémantique sont donnés ci-dessous :

```
void inverser(int* debut, int* fin);
```

Sémantique

Inverse le contenu du tableau 1D défini par *début* et *fin* où *début*, resp. *fin*, désigne l'adresse du premier, resp. du dernier, élément du tableau à inverser.

Exercice 11.13 Méli-mélo

Que va afficher le programme *main* ci-dessous ?
(On suppose travailler ici avec une architecture 32 bits)

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {

    int i = 1, j = 2;
    int *p = &i, *q = &j;

    *p = (int)q;
    *q = (int)p;
    p = (int*)(*p);
    q = (int*)(*q);

    i = 3, j = 4;
    printf("*p = %d q = %d\n", *p, *q);

    return EXIT_SUCCESS;
}
```

Exercice 11.14 Pointeurs, tableaux et fonctions

Que va afficher le programme *main* ci-dessous ?

```
#include <stdio.h>
#include <stdlib.h>

void f(int* p1, int p2);

int main(void) {

    int a = 5, b = 6;
    int t[2] = {3, 4};
    int *p = NULL, *q = NULL;

    f(&a, b);
    printf("1) %d, %d\n", a, b);

    p = &a;
    q = &b;
    f(q, *p);
    printf("2) %d, %d\n", *p, *q);

    f(t + 1, *t);
    printf("3) %d, %d\n", t[0], t[1]);

    f((int*)(&p), b);
    printf("4) %d, %d\n", a, b);

    return EXIT_SUCCESS;
}

void f(int* p1, int p2) {
    *p1 = 2 * p2;
}
```

Exercice 11.15* Utilisation de calloc

Ecrire une fonction C *initialiser* permettant 1°) de créer un tableau (à 1 dimension) de taille donnée, composé d'entiers (de type *int*), 2°) d'initialiser à une valeur donnée (de type *int*) tous les éléments de ce tableau.

IMPORTANT

La fonction :

- ne doit comporter que 2 paramètres : le premier fixant la taille (le nombre d'éléments) du tableau à créer, le second fixant la valeur à donner à tous les éléments du tableau
- doit utiliser les services de la fonction `calloc`

Ecrire aussi un petit programme de test permettant de vérifier le bon fonctionnement de votre fonction.

Exercice 11.16 Inversion d'un tableau 1D (2)

Implémenter la fonction C dont le prototype et la sémantique sont donnés ci-dessous :

```
int* inverser(const int* debut, const int* fin);
```

Sémantique

Renvoie le tableau inverse du tableau 1D défini par *début* et *fin* où *début*, resp. *fin*, désigne l'adresse du premier, resp. du dernier, élément du tableau à inverser.

Exercice 11.17 Adresses du min et du max d'un tableau 1D

Ecrire une fonction C qui prend en paramètre un tableau *tab* 1D de *int* de taille quelconque et qui renvoie en valeur de retour les adresses du plus petit élément et du plus grand élément¹ de *tab* (ou *NULL* si *tab* vaut *NULL* ou si *tab* est vide).

¹ Dans le cas où il y aurait plusieurs plus petits éléments on renverra l'adresse de celui dont l'indice dans *tab* est le plus petit (idem dans le cas du plus grand élément).

Exercice 11.18 Manipulation de la mémoire (1)

Ecrire un programme C qui :

- 1) déclare et initialise un tableau *tab1* à une dimension de *int*
- 2) copie (sans utiliser de boucle) le contenu de *tab1* dans un autre tableau *tab2*
- 3) affiche *tab1* et *tab2*
- 4) se termine

NB Prévoir une fonction dédiée à l'affichage d'un tableau sous la forme [valeur1, valeur2,...]. Ecrire cette fonction en utilisant exclusivement le formalisme pointeur.

Exercice 11.19 Manipulation de la mémoire (2)

Sans utiliser de boucle, compléter la partie notée *<à compléter>* dans le programme C ci-dessous de telle sorte que celui-ci affiche à l'exécution :

```
[0, 0, 0]
[1, 1, 1]
[2, 2, 2]
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SIZE 3

void afficher ... // Le code de la fonction afficher a volontairement été omis ici

int main(void) {

    int tab[SIZE] = {0};

    afficher(tab, SIZE);

    for (size_t i = 0; i < SIZE; ++i)
        tab[i]++;

    afficher(tab, SIZE);

    <à compléter>

    for (size_t i = 0; i < SIZE; ++i)
        tab[i] += 2;

    afficher(tab, SIZE);

    return EXIT_SUCCESS;
}
```

Exercice 11.20 Manipulation de la mémoire (3)

Que va afficher à l'exécution le programme C suivant ?

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
    char str[] = "memmove est tres utile.....";
    memmove(str + 17, str + 12, 10);
    printf("%s\n", str);
    return EXIT_SUCCESS;
}
```

Exercice 11.21 Initialisation d'une matrice

Ecrire une fonction C sans valeur de retour (*void*) qui initialise une matrice $m \times n$ de *int* avec des 1 sur les 4 "bords" et des 0 partout ailleurs.

Ecrire aussi un petit programme de test permettant de vérifier le bon fonctionnement de votre fonction.

Exercice 11.22 Somme des valeurs d'une matrice

Ecrire une fonction C qui renvoie la somme des coefficients de la matrice $m \times n$ de *double* passée en paramètre.

Ecrire aussi un petit programme de test permettant de vérifier le bon fonctionnement de votre fonction sur la matrice 3 x 4 suivante :

$$M = \begin{pmatrix} 1 & 2.5 & 3 & 4 \\ 5 & 6 & 7.5 & 8 \\ 9.5 & 10 & 11 & 12 \end{pmatrix}$$

Exercice 11.23 Diagonale d'une matrice carrée

Ecrire une fonction C qui prend en paramètre une matrice $n \times n$ de *int* et qui renvoie en valeur de retour le vecteur correspondant aux éléments de la diagonale gauche de ladite matrice.

Exemple : Si matrice = $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$, la fonction doit renvoyer le vecteur $\begin{pmatrix} 1 \\ 5 \\ 9 \end{pmatrix}$

Ecrire aussi un petit programme de test permettant de vérifier le bon fonctionnement de votre fonction.

Exercice 11.24 Calcul d'une intégrale

Ecrire une fonction C permettant d'intégrer, par la méthode des trapèzes, n'importe quelle fonction $f(x)$ (x de type *double*) entre deux bornes A et B (également de type *double*). Le nombre de pas d'intégration est aussi un paramètre de la fonction d'intégration.

Tester votre fonction d'intégration en écrivant un petit programme permettant de calculer :

$$\int_0^{\infty} e^{-x^2} dx \left(= \frac{\sqrt{\pi}}{2} = 0.886227 \right)$$

$$\int_0^{\frac{\pi}{2}} \sin(x) dx (= 1)$$

Chapitre 12 : Types composés

Exercice 12.1 Correction d'erreurs

Réécrire complètement le code ci-dessous de telle sorte à corriger tous les maladroites et erreurs qu'il contient.

Une fois dûment corrigé le programme devrait afficher à l'exécution :

a = 1, b = "ABC"

NB Aucune ligne de code ne doit être ajoutée ou supprimée.

```
#include <stdio.h>
#include <stdlib.h>

struct S {
    int a,
    char b[3]
}

void afficher(S* s);

int main(void) {
    S s = {a = 1; b = "ABC"};
    afficher(s) ;
    return EXIT_SUCCESS;
}

void afficher(S* s) {
    printf("a = %d, b = \"%s\"\n", s.a, s.b);
}
```

Exercice 12.2* Affichage d'une personne (1)

On suppose qu'une personne se caractérise par :

- son nom (20 caractères effectifs au maximum)
- sa taille en cm (type `uint8_t`)
- la couleur de ses yeux (type `enum`; couleurs possibles : bleu, vert, gris, marron, noir)

Ecrire un programme C qui, dans un premier temps, déclare, le plus proprement possible, tous les types nécessaires à la modélisation d'une personne (n'hésitez pas à faire usage de `typedef`) et qui, dans un second temps, déclare/définit deux fonctions (sans valeur de retour) `afficher_1` et `afficher_2`, permettant d'afficher l'ensemble des caractéristiques de la personne passée en paramètre. L'argument de la fonction `afficher_1` doit être passé par valeur, celui de `afficher_2`, par adresse.

Testez votre programme en considérant comme exemple de personne : Toto qui mesure 180 cm et qui a les yeux bleus.

Exercice 12.3 Affichage de plusieurs personnes

Essentiellement le même exercice que le précédent, mais on souhaite ici afficher non pas une personne mais plusieurs.

Plus précisément, il est demandé :

- de modéliser le concept de Personne en s'affranchissant de la contrainte des 20 caractères max pour le nom
- de déclarer un tableau contenant les 3 personnes suivantes :
 - Paul qui mesure 180 cm et qui a les yeux bleus
 - Pierre qui mesure 175 cm et qui a les yeux verts
 - Jean-Jacques qui mesure 182 cm et qui a les yeux marron
- d'écrire :
 - une fonction permettant l'affichage d'une personne (passée par adresse)
 - une fonction permettant l'affichage d'un tableau de personnes

Exemple d'exécution du programme :

Nom : Paul
Taille : 180
Yeux : bleus

Nom : Pierre
Taille : 175
Yeux : verts

Nom : Jean-Jacques
Taille : 182
Yeux : marron

Exercice 12.4 Lendemain d'une date

Ecrire un programme C qui détermine et affiche le lendemain d'une date donnée.

Exemple Le lendemain du 30.04.2016 est le 01.05.2016

Précisions

- Implémenter le concept de date sous la forme d'une structure à 3 membres : jour (de type `uint_8`), mois (de type `uint_8`) et année (de type `uint16_t`)
- Structurer le code en fonctions
- Tenir compte des années bissextiles dans le calcul du lendemain
- Afficher les dates au format `jj.mm.aaaa`

Exercice 12.5* Taille d'une structure

a) Soit la structure *S1* suivante :

```
struct S1 {  
    int* ptr;  
    int32_t a;  
    int64_t b;  
};
```

Quelle est la taille en bytes de *S1*

- 1) en architecture 32 bits ?
- 2) en architecture 64 bits ?

b) Soit la structure *S2* suivante :

```
struct S2 {  
    int8_t a;  
    int16_t b;  
    char c;  
    double* ptr;  
};
```

Quelle est la taille en bytes de *S2*

- 1) en architecture 32 bits ?
- 2) en architecture 64 bits ?

Exercice 12.6 Affichage d'une personne (2)

On souhaite modéliser des personnes, certaines de nationalité suisse, d'autres pas.

On suppose que toute personne (qu'elle soit suisse ou non) se caractérise par un nom (20 caractères effectifs max).

Pour les personnes de nationalité suisse, on souhaite (en plus du nom) aussi enregistrer le taux d'activité exercé (entier exprimé en %).

Pour les personnes de nationalité étrangère, on souhaite (en plus du nom) aussi enregistrer le type de permis de travail dont elle dispose (permis A, B ou C; à implémenter en tant que type énuméré).

Ecrire un programme C, qui, après avoir déclaré le plus proprement possible les types nécessaires à la résolution de notre problème, déclare/initialise les 2 personnes suivantes :

- "Toto", suisse travaillant à 80%
- "Titi", étranger avec permis C

... puis affiche à l'écran l'ensemble des caractéristiques de chacune de ces 2 personnes.

Exemple d'exécution du programme :

```
Nom           : Toto
Nationalite    : Suisse
Taux activite  : 80%
```

```
Nom           : Titi
Nationalite    : Etranger
Type permis    : C
```

Exercice 12.7 Bateaux

On souhaite modéliser des bateaux, plus précisément des voiliers et des bateaux à moteur.

On dispose des informations suivantes :

- Tout bateau a un nom (de longueur quelconque)
- Si le bateau est un voilier, il faut enregistrer la surface de la voilure en [m²] (type `uint16_t`) de celui-ci
- Les bateaux à moteur se divisent en 2 catégories : les bateaux de pêche et les bateaux de plaisance.
- Tout bateau à moteur se caractérise par la puissance totale de ses moteurs, exprimée en [CV] (type `uint16_t`)
- Si le bateau est un bateau de pêche, il faut enregistrer combien de tonnes de poisson celui-ci est autorisé à pêcher (de type `uint8_t`)
- Si le bateau est un bateau de plaisance, il faut enregistrer le nom de son propriétaire (de longueur quelconque)

Ecrire un programme C, qui, après avoir déclaré le plus proprement possible les types nécessaires à la résolution de notre problème, déclare/initialise les 3 bateaux suivants :

- "Alinghi", voilier ayant 300 [m²] de voilure
- "Espadon", bateau de pêche disposant de 2 moteurs de 500 [CV] chacun et pouvant pêcher 20 tonnes de poisson
- "Farniente", bateau de plaisance disposant d'un moteur de 100 [CV] et appartenant à Monsieur James Lamer

... puis affiche à l'écran l'ensemble des caractéristiques de chacun de ces 3 bateaux.

Exemple d'exécution du programme :

```
Nom           : Alinghi
Genre          : Voilier
Voilure [m2]   : 300
```

```
Nom           : Espadon
Genre          : Bateau de peche
Moteurs [CV]   : 1000
Pêche [t]      : 20
```

```
Nom           : Farniente
Genre          : Bateau de plaisance
Moteurs [CV]   : 100
Propriétaire   : James Lamer
```

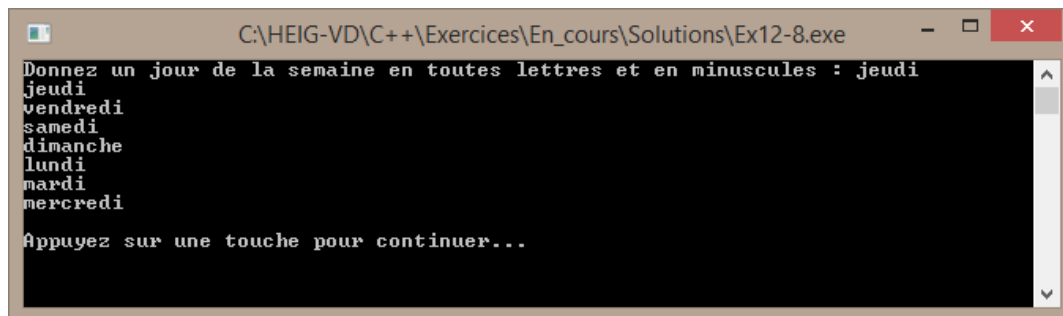
Exercice 12.8 Affichage des jours de la semaine (1)

En utilisant le concept de type énuméré, écrire un programme C qui :

- 1) demande à l'utilisateur de saisir, en toutes lettres et entièrement en minuscules, un jour de la semaine¹
- 2) affiche, entièrement en minuscules et à raison d'un nom par ligne, tous les jours de la semaine, en commençant par celui saisi par l'utilisateur
- 3) se termine

¹ Il n'est pas demandé ici de vérifier la saisie utilisateur.

Exemple d'exécution



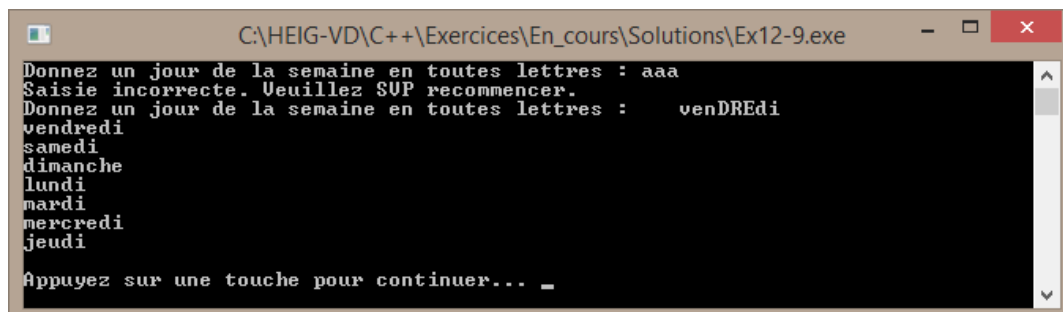
```
C:\HEIG-VD\C++\Exercices\En_cours\Solutions\Ex12-8.exe
Donnez un jour de la semaine en toutes lettres et en minuscules : jeudi
jeudi
vendredi
samedi
dimanche
lundi
mardi
mercredi
Appuyez sur une touche pour continuer...
```

Exercice 12.9 Affichage des jours de la semaine (2)

Même exercice que le 12.8, mais cette fois :

- l'utilisateur doit pouvoir saisir le nom d'un jour en minuscules et/ou majuscules avec d'éventuels espaces blancs devant et derrière
- le programme doit inviter l'utilisateur à refaire sa saisie si celle-ci n'est pas correcte

Exemple d'exécution



```
C:\HEIG-VD\C++\Exercices\En_cours\Solutions\Ex12-9.exe
Donnez un jour de la semaine en toutes lettres : aaa
Saisie incorrecte. Veuillez SUP recommencer.
Donnez un jour de la semaine en toutes lettres :  vendREdi
vendredi
samedi
dimanche
lundi
mardi
mercredi
jeudi
Appuyez sur une touche pour continuer... _
```

Exercice 12.10 Pile dynamique

Compléter la partie notée *<à compléter>* du code ci-dessous de telle sorte que celui-ci affiche (si l'utilisateur entre successivement les valeurs 1, 2, 3 et 0) :

```
Donnez un entier (0 pour terminer) : 1
Donnez un entier (0 pour terminer) : 2
Donnez un entier (0 pour terminer) : 3
Donnez un entier (0 pour terminer) : 0
```

```
3
2
1
```

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

<à compléter>

int main(void) {

    Pile pile = NULL;
    Info info;

    // Saisie utilisateur
    do {
        printf("Donnez un entier (0 pour terminer) : ");
        scanf("%d", &info);
        if (info != 0)
            empiler(&pile, info);
    } while (info != 0);

    // Affichage dans l'ordre inverse des valeurs saisies par l'utilisateur
    printf("\n");
    while (!estVide(pile)) {
        desempiler(&pile, &info);
        printf("%d\n", info);
    }

    return EXIT_SUCCESS;
}
```

Chapitre 13 : Chaînes de caractères

Exercice 13.1* Implémentation de *strlen*

Proposez une implémentation de la fonction *strlen* dont le prototype et la sémantique sont donnés ci-dessous :

```
size_t strlen(const char* s);
```

Sémantique :

Renvoie le nombre de caractères de la chaîne *s* (le `'\0'` final n'est pas comptabilisé)

Ecrire aussi un petit programme permettant de tester votre fonction.

Exercice 13.2 Implémentation de *strcpy*

Proposez une implémentation de la fonction *strcpy* dont le prototype et la sémantique sont donnés ci-dessous :

```
char* strcpy(char* to, const char* from);
```

Sémantique :

Recopie la chaîne *from* dans la chaîne *to* et retourne un pointeur sur *to*.

Ecrire aussi un petit programme permettant de tester votre fonction.

Exercice 13.3 Implémentation de *strncpy*

Proposez une implémentation de la fonction *strncpy* dont le prototype et la sémantique sont donnés ci-dessous :

```
char* strncpy(char* to, const char* from, size_t size);
```

Sémantique :

Recopie les *size* premiers caractères de la chaîne *from* dans la chaîne *to* et retourne un pointeur sur *to*. De manière plus précise :

- Si $size \leq strlen(from)$, *size* caractères sont copiés dans *to*.
Attention : Aucune marque de fin de chaîne (`'\0'`) n'est copiée dans *to*
- Si $size > strlen(from)$, $strlen(from)$ caractères ainsi que $size - strlen(from)$ `'\0'` sont copiés dans *to*.

Ecrire aussi un petit programme permettant de tester votre fonction.

Exercice 13.4 Implémentation de strcat

Proposez une implémentation de la fonction *strcat* dont le prototype et la sémantique sont donnés ci-dessous :

```
char* strcat(char* to, const char* from);
```

Sémantique :

Concatène la chaîne *from* à la suite de la chaîne *to* et retourne un pointeur sur *to*.

Ecrire aussi un petit programme permettant de tester votre fonction.

Exercice 13.5 Implémentation de strncat

Proposez une implémentation de la fonction *strncat* dont le prototype et la sémantique sont donnés ci-dessous :

```
char* strncat(char* to, const char* from, size_t size);
```

Sémantique :

Concatène la chaîne *from* à la suite de la chaîne *to* et retourne un pointeur sur *to*... mais au plus *size* caractères de la chaîne *from* sont copiés. Ici, contrairement à *strncpy* (voir exercice 13.3), un caractère '\0' est toujours ajouté à la fin de la chaîne (quel que soit le critère qui arrête la concaténation).

Ecrire aussi un petit programme permettant de tester votre fonction.

Exercice 13.6* Implémentation de strcmp

Proposez une implémentation de la fonction *strcmp* dont le prototype et la sémantique sont donnés ci-dessous :

```
int strcmp(const char* s, const char* t);
```

Sémantique :

Compare les deux chaîne *s* et *t* et retourne un résultat > 0 si $s > t$, $= 0$ si $s = t$ et < 0 si $s < t$.

Ecrire aussi un petit programme permettant de tester votre fonction.

Exercice 13.7 Implémentation de strchr

Proposez une implémentation de la fonction *strchr* dont le prototype et la sémantique sont donnés ci-dessous :

```
char* strchr(const char* s, int c);
```

Sémantique :

Retourne un pointeur sur la première occurrence du "caractère" *c* dans la chaîne *s* ou *NULL* si *c* n'est pas trouvé.

Ecrire aussi un petit programme permettant de tester votre fonction.

Exercice 13.8 Prénom et nom

Ecrire une fonction C qui prend en paramètres un prénom et un nom et qui retourne une nouvelle chaîne sous la forme "prénom nom".

Exemple : "James" + "Bond" => " James Bond".

Ecrire aussi un petit programme de test qui, après avoir demandé à l'utilisateur d'entrer un prénom, puis un nom, affiche à l'écran, en utilisant les services de la fonction ci-dessus :

La chaîne "prenom nom" comporte n caracteres.

Soit, dans notre exemple :

La chaîne "James Bond" comporte 10 caracteres.

Exercice 13.9 Inversion d'une chaîne

Proposer une implémentation pour chacune des fonctions d'inversion d'une chaîne suivantes :

1. **void** inverser_1(**char*** s);
2. **char*** inverser_2(**const char*** s);
NB renvoie s en cas de problème

Ecrire aussi un petit programme permettant de tester les deux fonctions.

Lister les avantages/inconvénients respectifs des 2 fonctions ci-dessus.

NB La fonction *strlen* (et elle seule !) de *stdlib* peut être utilisée ici

Chapitre 14 : Fichiers

Exercice 14.1* Lecture intégrale d'un fichier texte

On suppose disposer du fichier texte suivant :

```
1  
2  
3  
4  
5
```

Ecrire le plus proprement possible un programme C qui permette d'afficher à l'écran le contenu intégral de ce fichier.

Exercice 14.2 Ecriture d'un fichier binaire

Ecrire un programme C permettant de stocker dans un fichier binaire nommé *personnes.dat* des données relatives à des personnes.

On suppose qu'une personne se caractérise par les propriétés suivantes :

- son nom (20 caractères max)
- son prénom (15 caractères max)
- son âge (de type *unsigned short*)

Les données des diverses personnes doivent être saisies par l'utilisateur. On conviendra que la saisie se termine dès lors que l'utilisateur entre un nom de personne vide.

Exercice 14.3 Lecture intégrale d'un fichier binaire

On suppose disposer du fichier binaire *personnes.dat* créé dans l'exercice précédent.

Ecrire le plus proprement possible un programme C qui recopie le contenu intégral du fichier binaire *personnes.dat* dans un fichier texte *personnes.txt*, histoire de rendre les informations contenues dans le fichier binaire lisibles par un être humain.

Faire en sorte que les données du fichier texte apparaissent alignées en colonnes, comme suit :

Nom	Prenom	Age
Federer	Roger	34
Nadal	Rafael	30
Djokovic	Novak	29
del Potro	Juan Martin	27

Exercice 14.4 Recherche séquentielle dans un fichier binaire

Ecrire un programme C permettant, à partir du fichier *personnes.dat* créé dans l'exercice 14.2, de retrouver et d'afficher à l'écran les informations relatives à une personne de *nom* donné.

Exemples d'exécution

```
Quel nom recherchez-vous ? : Wawrinka  
Desole! Le nom "Wawrinka" ne figure pas dans le fichier
```

```
Quel nom recherchez-vous ? : del Potro  
Juan Martin del Potro, 27 ans
```

Exercice 14.5 Recherche par accès direct dans un fichier binaire

En exploitant l'accès direct, écrire un programme C permettant, à partir du fichier *personnes.dat* créé dans l'exercice 14.2, de retrouver et d'afficher à l'écran les informations relatives à une personne de *rang*¹ donné.

¹ Convention : La première personne stockée dans le fichier a pour rang 1, la deuxième, le rang 2, etc.

Exemples d'exécution

```
Quel rang recherchez-vous ? : 0  
Le fichier contient 4 enregistrement(s).  
Desole! Aucune personne ayant ce rang ne figure dans le fichier.
```

```
Quel rang recherchez-vous ? : 1  
Le fichier contient 4 enregistrement(s).  
La personne de rang 1 est : Roger Federer, 34 ans
```

```
Quel rang recherchez-vous ? : 4  
Le fichier contient 4 enregistrement(s).  
La personne de rang 4 est : Juan Martin del Potro, 27 ans
```

```
Quel rang recherchez-vous ? : 5  
Le fichier contient 4 enregistrement(s).  
Desole! Aucune personne ayant ce rang ne figure dans le fichier.
```

Exercice 14.6* Compteurs d'octets

Ecrire un programme complet permettant d'ouvrir n'importe quel fichier (image, son, document, ...) et de compter le nombre d'octets de chaque valeur (0 à 255) figurant dans ce fichier.

Le programme doit demander à l'utilisateur le nom du fichier (nom pouvant éventuellement contenir des espaces) à traiter. La saisie du nom doit être sécurisée (pas de débordement de buffer possible). Le nom du fichier (extension incluse) doit être plus petit ou égal à 30 caractères.

Avant de terminer le programme, il faut afficher le résultat à l'écran (compteurs de chaque valeur d'octet).

Exercice 14.7 Lecture d'un fichier texte sans utiliser de boucle

Ecrire un programme C permettant d'afficher à l'écran l'intégralité d'un fichier texte donné, sans jamais utiliser de boucle.