CHAPITRE 22

Exercices

Ce chapitre présente un certain nombre d'exercices dont les corrigés sont donnés sur un support électronique séparé[1]. Les exercices sont groupés par sous-chapitre, et la numérotation suit la numérotation des chapitres. Les numéros des exercices correspondent aux numéros des solutions.

Ces exercices ont été compilés avec différents compilateurs. La mention du ou des compilateurs n'est pas systématique. Il se peut que certains de ces exercices compilés avec une certaine version d'un compilateur puissent provoquer des erreurs sur certains autres. Il en va en particulier ainsi des exercices réalisables en C ANSI plutôt qu'exclusivement en C++. La compatibilité ANSI des compilateurs C++ est inégale, et souvent soumise au choix d'une ou plusieurs options pas toujours claires au premier abord.

L'ensemble des exercices corrigés a été réalisé en utilisant l'éditeur fourni avec le CD, soit Editeur V3.4. Il s'agit d'un logiciel "shareware", donc sujet à des droits d'utilisation. Ces droits doivent être payés à l'auteur, M. J.-P Menicucci, et se montent à environ 100 FF (vérifier le montant exact mentionné dans le formulaire d'enregistrement fourni dans l'aide du programme). Les coordonnées de l'auteur peuvent être trouvées à l'adresse indiquée sur la figure suivante. On peut aussi consulter le site (http://www.studioware.com) afin de vérifier s'il n'existe pas une version plus récente de ce logiciel. Le paiement des finances d'enregistrement vous donne accès à toutes les nouvelles versions de Editeur. Il est bien sûr possible aussi d'utiliser votre propre environnement de développement, au cas où vous auriez fait l'acquisition d'un autre système. Il est possible dans ce dernier cas de figure que certains caractères (minuscules accentuées, par exemple) ne soient pas supportées par votre environnement de développement.

L'éditeur supporte plusieurs formats de fichier, ainsi que trois langues de

1. Les corrigés des exercices font partie du CD de distribution du cours C++ ainsi que l'éditeur et le système de développement permettant la résolution de ces exercices.

dialogue (Français, anglais, italien). Il supporte également l'édition de macros, ce qui en fait un outil très efficace pour l'utilisateur averti. Sa simplicité d'utilisation en font un outil utilisable également par les novices. Enfin, la possibilité d' utiliser divers langages de programmation en font aussi un accessoire utile pour d'autres langages, comme Java, par exemple.

Le compilateur utilisé est également livré avec le CD : il s'agit d'un portage par Cygnus Solutions du compilateur de GNU pour Windows 32. Il est nécessaire de disposer de Windows 95 ou Windows NT pour utiliser cet environnement. Hormis le compilateur, qui compile aussi bien du code C ANSI (gcc) que C++ ANSI (g++), l'environnement fourni propose aussi divers utilitaires bien connus des utilisateurs de UNIX, comme make, flex, awk, etc... Les habitués de UNIX se trouveront à leur aise avec cet environnement qui inclut également le shell bash, alors que les habitués de Windows devront se réaccoutumer un peu à la manipulation de fenêtres alphanumériques. Cet exercice est de toutes façons nécessaire du fait que les exercices proposés utilisent intégralement des entrées-sorties alphanumériques. Le compilateur GNU est gratuit, et se trouve également sur les diverses moutures de UNIX, et bien sûr sur Linux. L'utilisation de ces exercices corrigés dans un autre environnement que Windows requiert une adaptation de format des fichiers. Consulter le site de Cygnus (http://www.cygnus.com) afin de vérifier s'il n'existe pas une nouvelle version : celle actuellement distribuée sur le CD est la version B20.

La difficulté des exercices est très variable. En principe, ils sont classés par ordre croissant de difficulté, sauf si la logique demande un autre séquencement (exercices liés à la terminaison préalable d'une autre exercice). La difficulté de l'exercice est signalée par un code alphanumérique dans la donnée. Ce code correspond aux indications de difficulté fréquemment rencontrées pour la cotation d'itinéraires de montagne par l'UIAA (Union Internationale des Alpinistes Amateurs); ceci ne signifie pas que les dangers inhérents à la réalisation de l'exercice soient à la mesure du danger rencontré dans un itinéraire de montagne correspondant ! Le plus grave danger auquel on s'expose dans ce genre d'exercice est une éventuelle frustration :

- F : facile. Réalisable en quelques minutes.
- PD : peu difficile. Réalisable en une demi-heure.

- AD : assez difficile. Il vaut mieux prévoir une petite heure.
- D : difficile. Un après-midi (4 heures) devrait néanmoins suffire.
- TD : très difficile. Une journée (8 heures) sera bien remplie.
- ED : extrêmement difficile. Il s'agit d'un projet de longue haleine, et le corrigé n'est pas forcément fourni, parceque l'auteur n'a peut-être pas trouvé de solution satisfaisante.
- XD : exceptionnellement difficile. Niveau travail de doctorat. Fourni sans corrigé.

Les temps indiqués peuvent varier pour certains exercices. En réalité, certains exercices de niveau facile contiennent un nombre élevé de points à traiter, ce qui peut effectivement prendre pas mal de temps.

Les corrigés, quand ils existent, sont regroupés par dossiers. Chaque sous-chapitre de cette liste d'exercices a un dossier correspondant localisé à l'intérieur du dossier "Exercices corrigés". Le nom du dossier contenant les corrigés est mentionné dans l'en-tête des données d'exercices, et les fichiers respectifs sont indiqués dans la donnée.

22.4 *Une rapide introduction à C++*

Les corrigés se trouvent dans le dossier **Exercices corrigés/Hello.**

1. (F) Ecrire un programme qui écrit “Hello” sur la console en utilisant `<stdio.h>`
2. (F) Ecrire un programme qui écrit “Hello” sur la console en utilisant `<iostream.h>`.
3. (F) Ecrire un programme qui permet de lire un entier et un caractère du clavier en utilisant `<stdio.h>`.
4. (F) Ecrire un programme qui permet de lire un entier et un caractère du clavier en utilisant `<iostream.h>`

22.5 *Le préprocesseur*

Les corrigés se trouvent dans le dossier **Exercices corrigés/Le préprocesseur.**

1. (F) Ecrire un programme affichant systématiquement le numéro de la ligne qu'il est en train d'exécuter (Corrigé : **LineNo.c**).
2. (F) Ecrire un programme utilisant soit stdio.h soit iostream.h selon qu'il est compilé par un compilateur C ou un compilateur C++.

22.6 *Types de base et dérivés*

Tous ces exercices ne sont pas solubles sans un petit “regard en avant” sur les fonctions. Les notions introduites dans le paragraphe définissant “C++ en un clin d’oeil” devraient pourtant être suffisantes. Les corrigés se trouvent dans le répertoire **Types**.

1. (F) Ecrire une macro définissant le type `Byte`. (Un byte est un mot de huit bit, dont le MSB (*Most Significant Bit*) représente le signe). On suppose que la machine cible utilise un jeu de caractères de type USASCII.
2. (PD) Définir une structure implémentant des variables VeryLongInt (entiers sur 256 bit).
3. (PD) Afficher sur l’écran une variable de type VeryLongInt.
4. (AD) Lire depuis le clavier une valeur de type VeryLongInt.
5. (F) Définir un type VeryLongInt.
6. (PD) Utiliser le code défini en (exercice 3, page342) et (exercice4, page342) en l’incluant à la structure définissant le type VeryLongInt.

22.7 *Types standard introduits par C++*

22.8 *Opérateurs standard et instructions*

Avant d'aborder ces exercices, il est nécessaire d'étudier également le chapitre consacré aux instructions. Pour faire le moindre exercice ayant un intérêt quelconque, il faut avoir une connaissance au moins élémentaire des opérateurs de base et des instructions du langage. Les corrigés de ces exercices peuvent être consultés dans le dossier **"Exercices corrigés/Opérateurs standard"**.

1. (PD) Ecrire un programme permettant l'entrée d'un nombre entier depuis le clavier, et l'affichage de :

- Son carré
- Sa factorielle
- Les bits le composant (prendre garde à la taille de l'entier pouvant varier selon les implémentations)
- Sa parité
- S'il s'agit d'un carré parfait, et si oui, afficher la racine.
- S'il s'agit d'un nombre premier

Le programme bouclera sur lui-même tant qu'on ne lui a pas donné la valeur 0 à traiter.

La sortie du programme réalisé ressemblera à peu près à la séquence suivante (Metrowerks Code Warrior avec des entiers de 2 byte) :

```
Entrer un entier
13

Carre de 13 : 169
La factorielle de 13 est -13312
Representation binaire de 13: 000001101
13 est un nombre impair
13 n'est pas un carre parfait
13 est un nombre premier
Entrer un entier
5

Carre de 5 : 25
La factorielle de 5 est 120
Representation binaire de 5: 000000101
5 est un nombre impair
5 n'est pas un carre parfait
5 est un nombre premier
Entrer un entier
16

Carre de 16 : 256
La factorielle de 16 est -32768
Representation binaire de 16: 000010000
16 est un nombre pair
16 est un carre parfait, dont la racine vaut 4
16 n'est pas premier, puisque divisible par 2
Entrer un entier
```

```
0

Au plaisir de vous revoir ...
```

Corrigé : **BasicOps.cpp**.

2. (PD) Ecrire un programme permettant l'introduction d'un réel et d'un entier au clavier, et affichant le réel élevé à la puissance entière. Chercher à optimiser l'algorithme en minimisant le nombre de multiplications réelles. Le programme bouclera sur lui-même tant qu'on ne lui a pas donné la valeur 0 comme puissance entière à traiter. Corrigé : **Power.cpp**.

22.9 *Instructions*

Les exercices spécifiques aux instructions (chapitre 9) sont regroupés avec les exercices spécifiques aux opérateurs standard (Voir *Opérateurs standard et instructions*, page 344.), ces deux thèmes étant indissolublement liés lorsque l'on veut écrire ne serait-ce que le plus élémentaire des programmes.

22.10 *Fonctions*

Les exercices portant sur l'utilisation de fonctions sont à implémenter sous forme de modules séparés, comme il est d'usage en C. La fonction fait l'objet d'un fichier de définition (X.h) importable (`#include`) par un autre programme, et d'un fichier implémentation (X.C, X.cp, X.cpp) compilé séparément, et lié au programme appelant par l›éditeur de liens (*linker*), sous forme de code objet. Les corrigés de ces exercices se trouvent dans le dossier "**Exercices corrigés/Fonctions**"

1. (F) Reprendre l'exercice exercice1, page344 sur les opérateurs standard, et convertir la série d'opérations implémentée en une série d'appels de fonctions. Les fonctions seront implémentées dans un module de compilation séparé, doté d'un fichier interface.
2. (PD) Ecrire une fonction calculant le nombre de Fibonacci d'un nombre passé en paramètre. Rappelons que le nombre de Fibonacci F(n) est défini comme suit :

   F(0) = 1;

   F(1) = 1;

   F(n) = F(n - 1) + F(n-2)

   Comparer une solution récursive et une solution non récursive; critiquer le résultat. Peut-on en tirer des conclusions générales ?

   Corrigé : **Fibonacci/fibo.cpp**

3. (PD) Ecrire une fonction implémentant le tri de valeurs entières. Cette fonction devra être appelable aussi bien avec un tableau constant qu'un tableau non constant. La méthode de tri utilisée (quicksort, bubble, insertion, etc...) est laissée au libre choix de l'implémentateur. Corrigé : **Quicksort/main.cpp**
4. (PD) Ecrire une fonction calculant le plus grand diviseur commun (PGCD) de deux nombres entiers. Corrigé : **Pgcd/Pgcd.cpp**
5. (AD) Dans un système de télécommunications, on peut estimer les pertes dans un réseau de connexion en utilisant la relation d'Erlang :

   Cette relation a l'allure suivante : $B = \dfrac{\dfrac{A^N}{N!}}{\sum_{i=0}^{N} \dfrac{A^i}{i!}}$

   A est le **trafic**, et mesure le degré d'occupation des sources. C'est le produit du taux de sollicitation de ces sources par la durée d'occupation par les serveurs. Ainsi, si il vient en moyenne un client par heure à un guichet, et que le guichetier met en moyenne 10 minutes pour traiter le client, ce guichetier va traiter un trafic de 1/6 Erlang.

   N est le **nombre de serveurs** (ou le nombre de guichetiers, si l'on préfère)

B est la **probabilité** pour une sollicitation de ne pas trouver de serveur (**probabilité de pertes**).

Complexe à évaluer, cette relation se prête bien au calcul par ordinateur. On désire fixer son calcul dans un module de librairie, dont on pourra importer le code au besoin. Ecrire une fonction importable par un programme quelconque permettant l'évaluation de la probabilité de pertes selon Erlang, puis écrire un programme de test utilisant cette fonction. La sortie du programme de test ressemblera à peu près à la sortie suivante :

```
Entrer le trafic
10
Entrer le nombre de serveurs devant traiter 10 Erlang
20
Probabilite de pertes pour un trafic de 10 Erlang sur 20 serveurs
     B = 0.186913 %

Entrer le trafic
100
Entrer le nombre de serveurs devant traiter 100 Erlang
150
Probabilite de pertes pour un trafic de 100 Erlang sur 150 serveurs
     B = 6.51117e-05 %

Entrer le trafic
0


Trafic nul ou negatif, bye, bye ...
```

On cherchera à réaliser un programme capable de calculer cette relation pour un grand nombre de serveurs (ordre de grandeur 10000)

Corrigé : **Erlang/Erlang.cpp**

6. (D) **Liste générique**. Implémenter une série de fonctions permettant le contrôle d'une liste chaînée (linked list). Cette liste permettra de stocker n'importe quel type de données, mais sera homogène (tous les éléments de la liste stockent des données identiques). Les possibilités offertes par cette liste seront :

- Création d'une liste
- Destruction d'une liste
- Parcours de la liste avec appel d'une fonction définie par l'utilisateur pour chacun des éléments de la liste
- Insertion à une position déterminée de la liste
- Destruction d'un élément
- Ajout d'un élément en queue de liste

On veillera dans cette implémentation à ce que le code écrit puisse implémenter plusieurs listes différentes simultanément. En d'autres termes, il est préférable d'éviter les données statiques dans le module d'implémentation.

7. (AD) **Liste générique (suite)**. Est-il possible d'étendre la liste de l'exercice6, page348 à une liste inhomogène (c'est-à-dire contenant des données pas forcément de même type) ? Si c'est possible, est-ce judicieux ?

22.11 *Opérateurs*

Il s'agit ici plus particulièrement des opérateurs spécifiques à C++, ainsi que de constructions plus évoluées du langage C. En particulier, on découvrira dans ces exercices, quelques exercices plus spécifiquement dévolus aux pièges que peuvent receler les pointeurs, et la manière d'utiliser des références pour accroître la sécurité d'un programme.

Les corrigés se trouvent dans le classeur **Exercices corrigés/Opérateurs**.

1. (AD) Soit le type suivant :

```
typedef   struct
     {
     double    real;
     double    imag;
     }    Complex;
```

   Définir les opérateurs +, - , / et * pour le type Complex et les intégrer dans un module de compilation séparé du programme de test, avec un fichier de définition de prototypes (.h). Le corrigé de cet exercice ainsi que du suivant se trouve dans le dossier **Complex *f*:complex.cp / complex.h**.

2. (F) Pour l'exercice précédent, on désire disposer d'un **opérateur** << agissant sur un ostream, et d'un **opérateur** >> agissant sur un istream, de manière similaire aux opérateurs correspondant pour les types standard.

3. (PD) Ecrire un petit programme créant **dynamiquement** un tableau d'entiers en mémoire, de dimensions spécifiées lors de l'exécution, par une entrée de l'utilisateur. Initialiser le tableau ainsi crée à l'aide de la fonction de librairie rand() (Utiliser l'aide en ligne du système de développement pour localiser rand()). Utiliser ensuite la fonction de tri développée lors de l' exercice3, page347 pour trier ces valeurs. Avant la terminaison du programme, veiller à libérer la place occupée par le tableau.

22.12 *Classes*

Les corrigés de ces exercices se trouvent dans le classeur **Exercices corrigés/Classes**.

1. (F) **Reprendre et utiliser la classe Strings** définie dans le cadre de ce chapitre avec un programme de test**.**
2. (F) **Utiliser la classe Point** définie dans ce chapitre dans un programme principal. Le programme principal inclura le fichier Point.h, et lors de l'édition de liens, utilisera le fichier résultant de la compilation de Point.C. Le programme principal (main.C) se contentera d'instancier quelques objets de type Point, et d'afficher leur contenu. Corrigé: **Point:Point.cp**
3. (F) Ajouter la méthode `move(float deltax, float deltay)` à la classe Point, et la tester.
4. (AD) **Soit la classe suivante** :

```
#include <iostream.h>
class X {
     int i;
     public :
          void setI(int k) { i = k;}
          int  getI() { cout<<i<<endl; return i; }
     };
```

- i peut-il être modifié directement par un programme principal instanciant la classe X ?
- Vérifier l'existence et le fonctionnement du constructeur par défaut.
- Vérifier l'existence et le fonctionnement du constructeur de copie.
- Ecrire un constructeur de manière à interdire à un utilisateur l'instanciation de cette classe sans valeur initiale. Utiliser le code du constructeur pour ajuster i à la bonne valeur.
- Modifier la classe X de manière à ce que le membre privé i soit une référence, et non plus une valeur, comme dans :

```
class X   {
               int& i;// Référence, et non plus valeur !
          public :
               .....
          };
```

- Comment modifier les membres publics de la classe pour que le programme compile et fonctionne? Pourquoi?
- Ecrire un constructeur de copie, et s'assurer que le constructeur de copie generé jusque là par le compilateur n'est plus utilisé.

5. (AD) **Convertir le module développé** au cours de l'exercice 1, page350 implémentant des opérations sur des nombres complexes, en une classe Complex.
6. (D) **Tableau de dimensions variables**.

   Sous C (et C++), les tableaux sont implémentés par la syntaxe suivante :

```
<type>  <dénomination_du_tableau>[<dimension>];
```

Exemple :

```
int  tableauEntier[200];
```

déclare un tableau de 200 entiers.

Cette implémentation souffre des limitations et des inconvénients suivants :

- L'indication de dimension ne fait pas partie du tableau. Lors du passage d'un tableau comme paramètre à une procédure, il est nécessaire de passer la dimension du tableau de manière séparée à la procédure invoquée (ou pire, de stocker cette information dans une variable accessible globalement).
- La dimension du tableau est fixe. Il n'est pas possible de faire varier la taille d'un tableau en fonction de ses besoins, si bien qu'une application doit soit déclarer systématiquement des dimensions maximales (comment les estimer ?), soit utiliser des pointeurs avec les risques que l'on peut imaginer.
- La syntaxe utilisée pour détruire (delete []) un tableau déclaré à l'aide de pointeurs est différente de celle utilisée pour détruire un élément simple (delete), ce qui augmente le risque d'erreurs.
- Du fait que les dimensions du tableau sont fixes, il n'est pas possible d'optimiser la place mémoire occupée.
- La borne inférieure du tableau est toujours 0. Il n'est pas possible de déclarer, de manière analogue à PASCAL, par exemple, un tableau sous la forme ARRAY[-5..12] OF integer ;
- On ne peut pas copier un tableau dans un autre sans utiliser une boucle itérative fastidieuse, inversément à des langages comme PASCAL ou MODULA-2.
- Un tableau n'offre pas de sécurité d'accès. Toute tentative d'indexer (par indexation ou à l'aide d'un pointeur) un élément inexistant est, au mieux, détectée par une violation de la protection mémoire, quand cette dernière existe (c'est heureusement le cas sous UNIX). Au pire, on accède à d'autres variables placées à proximité du tableau, ce qui promet un diagnostic de pannes assez ardu.
- Dans le même ordre d'idées, la possibilité d'accéder aux éléments d'un tableau à l'aide de pointeurs augmente le risque d'accès illicites à l'aide de pointeurs non correctement initialisés.
- On ne peut pas simplement écrire un tableau dans un flot d'entrées-sorties.

On se propose de développer un type tableau éliminant tout ou partie des inconvénients cités çi-dessus. Pour simplifier, dans un premier temps, nous ne considérons que le type tableau d'entiers. Nous nous réserverons une éventuelle généralisation pour des exercices futurs. A chacun d'écrire son propre cahier des charges du tableau qu'il désire implémenter, en étant entendu qu'il devra l'implémenter lors des exercices. La liste d'inconvénients çi-dessus n'est pas exhaustive, et certains de ces inconvénients peuvent ne pas être ressentis comme tels par un implémentateur donné. Libre donc à chacun de définir le tableau qui lui paraît le mieux correspondre à ses désirs d'utilisateur. Le corrigé se trouve dans le dossier **IntArray:IntArray.cp**.

7. (D) **Sémaphores et mémoire partagée sous UNIX**

On se propose d'encapsuler le mécanisme de sémaphores de UNIX, ainsi que les mécanismes de mémoire partagée, dans une classe C++. Ces mécanismes comprennent les queues de messages (*Message queues*), les sémaphores (*Semaphores*), et les zones mémoire partagées (*Shared Memory*). Le corrigé se trouve dans le dossier **IPC:MSGQ.C / IPC:MSGQ.H** et **IPC:SMC.C / IPC:SMC.H**.

8. (D) **Liste générique**.

   Mettre les fonctions développées lors de l'exercice6, page348, sous forme de classe.

22.13 *Classes imbriquées*

22.14 *Héritage*

1. (AD) Soit les deux classes suivantes :

```
class A {
     protected :
          int i;
     public :
          A(int k) : i(k) { cout<<"A constructed "<<i<<"\n"; }
          ~A()  { cout<<"A deleted\n"; }
     };

class B : public A {
     private :
          int j;
     public :
          B(int k, n = 1) : A(k), j(n)
               { cout<<"B constructed "<<j<<"\n"; }
          ~B()  { cout<<"B deleted\n"; }
     };
```

- Que génère le programme suivant ?

```
main() { A a(1); B b(2); }
```

  Vérifier sur les machines et commenter.

- Que génère le programme suivant ?

```
int  main(char**, int)
     {
     A *a, *b;
     a = new A(10);
     b = new B(20, 30);
     delete a;
     delete b;
     return 0;
     }
```

  Vérifier sur les machines et commenter.

  Est-ce correct ? Si non, localiser et corriger l'erreur

- Définir, pour les deux classes, la méthode `print()` qui imprime la valeur des membres privés ou protégés. Pour les deux classes, cette fonction doit avoir la même signature. Définir une procédure `printA(*A)` [`void printA(A *theArgument)`] dont l'argument est un *A, et qui invoque simplement la méthode `print()` de `theArgument`. Que génère le programme suivant :

```
int  main(char**, int)
     {
     A *a, *b;
     a = new A(10);
     b = new B(20,30);
     printA(a);
     printA(b);
```

```
        delete a;
        delete b;
        return 0;
        }
```

Vérifier sur les machines et commenter.

- Que faudrait-il pour que la méthode `print` de B imprime également la valeur de A ?
- Soit le programme suivant :

```
  int  main(char**, int)
       {
       A *a, *b;
       a = new A(10);
       b = new B(20, 30);
       B b1(*b);
       delete a;
       delete b;
       return 0;
       }
```

Commenter le résultat. Peut-on faire en sorte que ce programme fonctionne correctement ?

- Pour le programme précédent, remplacer la construction par copie de b par une copie de a, de manière semblable à B b2(*a); en ayant tenu compte des constatations faites lors de la question précédente. Que se passe-t-il ? Commenter le résultat.
- Définir une classe C qui contient un objet de classe B comme membre privé, et tracer également la construction/destruction. Comment se comporte l'instanciation de la classe C ?
- Vérifier le comportement du constructeur de copie généré automatiquement pour A, B et C.

2. (F) Quand faut-il impérativement déclarer un destructeur comme virtuel ? Peut-on émettre une règle généralement applicable ?
3. (D) Implémenter une hiérarchie d'objets graphiques, parmi lesquels on introduira le point, la droite, l'ellipse, le cercle, le rectangle, le carré, un polygone quelconque, etc.... Les surfaces peuvent être remplies, les traits peuvent avoir diverses épaisseurs. Pour éviter le recours à un système de dessin (Macintosh, Windows ou X), on implémentera les méthodes de dessin par de simples messages sur un terminal (Exemple : je suis un carré de côté XX centré en YY). Le programme de test devra être en mesure de générer dynamiquement des objets graphiques sur demande de l'utilisateur et les "dessiner" sur le terminal.

22.15 *Héritage multiple*

Soit la classe de base suivante :

```
class   X
    {
    int     i;
    public :
        X() { cout<<"Default constructor"<<endl; }
        X(int x) : i(x) { cout<<"X Constructed "<<i<<endl; }
        virtual ~X() { cout<<"X (~X) Destructed "<<i<<endl; }
        virtual void    print()
            {
            cout<<"print X called "<<i<<endl;
            }
    };
```

Soit les classes derivées suivantes :

```
class   A : public X
{
    int     i;
    public :
        A(int k) : i(k) { cout<<"A Constructed "<<i<<endl; }
        virtual ~A() { cout<<"A (~A) Destructed "<<i<<endl; }
        virtual void    print()
            {
            cout<<"print A called "<<i<<endl;
            X::print();
            }
    };

class   B : public X
{
    int     j;
    public :
        B(int n) : j(n) { cout<<"B Constructed "<<n<<endl; }
        virtual ~B() { cout<<"B (~B) Destructed "<<j<<endl; }
        virtual void    print()
            {
            cout<<"print B called "<<j<<endl;
            X::print();
            }
    };

class   C : public A, public B
    {
    int     j;
    public :
        C(int k, int a, int b, int x) : A(a), B(b), X(x), j(k)

            { cout<<"C Constructed "<<j<<endl; }
        virtual ~C() { cout<<"C (~C) Destructed "<<j<<endl; }
        virtual void    print()
            {
```

```
            cout<<"print C called "<<j<<endl;
            B::print();
            A::print();
            }
    };
```

puis le programme principal suivant :

```
main()

    {
    cout<<"Constructor"<<endl;
    C   cc(100, 200, 300, 123456);
    cc.print();
    }
```

1. Quel est le problème ? Comment le corriger ?

- Soit la procédure

```
void      aProc(C anInstance)

     {
     anInstance.print();
     }
```

que se passe-t-il si on appelle cette procédure depuis le programme principal? Peut-on contrôler ce phénomène ?

2. Créer une classe Motor et une classe Vehicle avec constructeurs et destructeurs, et une méthode print qui permette d'identifier les instances (par exemple avec un entier comme membre privé, ou une chaîne de caractères quelconque. Faire dériver de ces deux classes la classe MotorVehicle (avec également une méthode print) et observer le comportement lors de l'instanciation. Définir une procédure p prenant comme paramètre soit un pointeur sur un Motor ou sur un Vehicle, et lui passer un pointeur sur un MotorVehicle.

3. Reprendre l'exercice précédent, et définir la classe Engine, dont dérivent à la fois Motor et Vehicle. Modifier la définiton des classes en conséquence, et modifier la procédure p pour qu'elle accepte comme paramètre un pointeur sur des Engine. Observer et commenter les comportements :

   a) dans le cas où Engine est une classe de base virtuelle (class Motor : virtual public Engine {} ... class Vehicle : virtual public Engine {}.. etc...)

   b) dans le cas où Engine est une classe de base normale (class Motor : public Engine .. class Vehicle : public Engine {}..... etc...)

   c) dans le cas où Engine est une classe de base normale et où la classe MotorVehicle n'implémente pas de méthode print() (class Motor : public Engine .. class Vehicle : public Engine {}..... etc...).

   d) Comment appeler la méthode print d'un Vehicle ou d'un Motor faisant partie d'une instance de MotorVehicle?

22.16 *Templates*

1. (PD) Implémenter la classe Array avec les mêmes fonctionnalités que la classe définissant les tableaux dynamiques (paragraphe6, page351) pour des entiers, mais cette fois pour des types de données quelconques. (Voir aussi l'exemple donné dans le cours).
2. (AD) Implémenter une fonction template opérant le tri d'un tableau (paragraphe3, page347) de valeurs de type quelconque. Quelles exigences poserez-vous au type de données à trier ?
3. (PD) Reprendre la classe Array définie en exemple dans ce chapitre (paragraphe 1, page359). Y ajouter une méthode sort() (paragraphe2, page359) permettant le tri du tableau, et un opérateur <<, permettant l'impression du tableau sur un ostream. Ces méthodes additionnelles posent-elles des exigences particulières aux variables stockées dans le tableau? Lesquelles? Critiquer l'implémentation.

22.17 *Classes génériques*

1. (AD) Dans le cadre d'un projet, il est nécessaire d'implémenter une file d'attente simple, de type FIFO (First In First Out), de longueur infinie, contenant des instances d'une classe A donnée. Définir une classe générique permettant d'implémenter le modèle général associé à une file d'attente de ce type, puis en déduire une implémentation spécifique au problème de la classe A. (Note : il est judicieux de faire au préalable un effort de généralisation)
2. (AD) Plus tard, dans un autre projet, on désire implémenter à nouveau une file d'attente, mais de taille limitée, en introduisant des priorités (certains membres peuvent être traités en priorité). Les objets ne pouvant pas être mis en file d'attente sont perdus, mais l'utilisateur de la file d'attente est averti de cette perte, de manière à pouvoir entreprendre des mesures adéquates en cas de débordement. Montrer que le code du projet précédent peut être réutilisé.
3. (D) Dans un troisième projet, on désirerait ajouter à la file précédente la notion de temps d'attente maximum limité, ainsi que des priorités variables (A chaque attente supplémentaire, la priorité augmente). Un élément qui excède son temps d'attente maximum est soit perdu, soit signifié à l'utilisateur au moyen d'une fonction (callback function) qu'il définit lui-même. Montrer que le code du projet précédent peut être réutilisé.
4. (D) Peut-on sans autre implémenter une file d'attente de type LIFO (Least In, First Out) à partir du code généré pour les précédents projets? (Note : une file de type LIFO est, dans son expression la plus simple, identique à une pile).
5. (F) Implémenter le fichier typé décrit dans le cadre du chapitre décrivant les template.
6. (D) Un fichier cyclique est un fichier qui se surécrit lorsqu'il atteint une taille déterminée. Implémenter un fichier cyclique à l'aide d'une classe générique, et en dériver un template pour l'utilisateur final.
7. (D) Implémenter une classe générique implémentant le type fichier comprimé, en utilisant une méthode de compression à choix (par exemple Huffmann), et en dériver un template pour l'utilisateur final. Peut-on faire en sorte que l'utilisateur puisse choisir parmi diverses méthodes de compression / décompression, voire au besoin implémenter sa propre méthode ?

22.18 *Exceptions*

Les corrigés de ces exercices se trouvent dans le classeur **Exercices corrigés:Exceptions**.

1. (F) Sur la base du petit exemple présenté dans le cours, écrire un exemple de traitement d'exceptions basé sur la valeur d'une variable globale. Corrigé : **except.cp**.
2. (F) Reprendre l'exercice précédent, et vérifier si le compilateur que vous utilisez traite correctement la destruction des objets en situation d'exceptions. La réponse peut évidemment varier d'un compilateur à l'autre... Corrigé : **except.cp**. Note : le corrigé ne fournit pas la réponse, mais la manière de l'obtenir...