

Vincent Castille
Mathieu Lapeyre

22 Mars 2005

Elèves du Groupe 3

Projet tuteuré C++

Sous la direction de P.Sebilo

Sommaire

I. Enoncé Complet

A. Mise en œuvre :

L'objectif de ce projet est de réaliser un travail de synthèse et de révision d'un maximum de notions élémentaires que nous avons abordées en cours de manière dispersée. La plupart des algorithmes mis en œuvre ont déjà été résolus en cours.

Ce projet pourra être réalisé en solo ou en binôme. Bien entendu, je serai plus exigeant pour les étudiants qui travaillent en binôme.

Il est fortement recommandé de vous mettre au travail le plus rapidement possible et de ne pas attendre le dernier moment pour rendre votre dossier.

Ce projet comporte trois parties qui devront être traitées successivement :

- Définition des classes dont les déclarations sont imposées.
- Utilisation de ces classes dans 3 programmes de mise à jour d'un fichier.
- Complémentation des classes sans directives précises

Vous remarquerez que pour simplifier, les exceptions ne sont pas volontairement traitées. On vous demandera de les ajouter dans un deuxième temps. Même remarque pour les fichiers inclus.

Il est fortement conseillé de tester les classes avant de les utiliser dans les exercices de la fin. Pour cela, vous devez écrire pour chacune des classes un programme qui teste l'ensemble des fonctions avec un ensemble de paramètres choisis de la manière la plus exhaustive possible. A titre d'exemple, je vous fournis mon programme de test de la classe CNom que vous pouvez utiliser pour tester votre propre classe .(fichier TNom.cxx) .

Ce travail débouchera sur un dossier papier et un dossier informatique :

- Le dossier papier comportera les listings demandés ainsi que les explications nécessaires à la compréhension des programmes. Il devra être relié et comporter sur la première page toutes les informations indispensables pour pouvoir identifier les auteurs et la nature du sujet traité. L'analyse de ce dossier, pourra être éventuellement complété par un oral où vous serez amenés à me fournir toutes explications supplémentaires sur les méthodes choisies ainsi que sur votre travail personnel.
- Le dossier informatique devra comporter ces mêmes programmes stockés dans un répertoire nommé cppps créé directement dans votre répertoire racine. Ces programmes devront pouvoir être testés avec mes propres programmes de test. Pour que cela soit possible, vous ne devez pas modifier les fichiers de déclarations qui vous sont fournis.

Les deux dossiers devront être rendus au plus tard le **Vendredi 19 Mars 2005** au secrétariat. Tout retard sera sanctionné au niveau de la note.

Pensez à bien documenter vos listings et à soigner la présentation, j'en tiendrai le plus grand compte dans la note.

Des informations supplémentaires pourront être fournies par mail.

Bien entendu, vous pouvez me consulter pour toute demande d'explications supplémentaires.

B. Première partie : Les classes :

1) Classe CString (Listings à fournir : CString.h et CString.cxx) :

Nous avons déjà étudié en cours une version réduite de la classe CString qui dérive de la classe std::string. Nous devons compléter cette classe pour traiter les questions qui vont suivre :

Question 1 :

Ajouter la fonction non membre :
string UnsignedToString (const unsigned i) ;
qui transforme un unsigned decimal en string.

Question 2 :

Ajouter la fonction non membre :
unsigned StringToUnsigned (const string & Str) ;
qui transforme une chaîne supposée contenir l'image d'un unsigned décimal ;
(utiliser l'algorithme d'Horner)

Question 3 :

Procédure de lecture contrôlée :
En cours, nous avons réalisé une procédure de lecture 'à la volée'.
Nous allons compléter cette procédure en contrôlant le type des caractères saisis.

Pour cela, nous devons définir une procédure où la fonction qui valide les caractères saisis est passée en paramètre. Lorsqu'on passe une fonction en paramètre, on passe en réalité un pointeur sur la fonction dont l'identificateur est le nom de cette fonction. Le problème c'est qu'il n'existe pas de type de pointeur standard sur les fonctions. Le type du pointeur dépend du type de la fonction, du

nombre et du type de ses paramètres. On vous propose donc les déclarations suivantes :

```
typedef bool (* FctValidChar_t) (const char) ;  
// FctValidChar_t est donc le type pointeur de fonction de type booleen (ou  
prédicat) qui ont un paramètre donné de type char.
```

```
Void LireString ( FctValidChar_t isGood, const bool AvecEcho = true) ;  
// Fonction membre de la classe Cstring  
// Les caractères non conformes sont refusés (avec sonnerie)  
// Possibilité de retour en arrière mais pas au - delà de la première position.  
// Fin de saisie exclusivement sur <cr>
```

Exemple d'utilisation (pour lire une chaîne en majuscules) :

```
CString MaStr ;  
MaStr.LireString (isupper) ;
```

2) Classe CNom (Listing à fournir : CNom.cxx) :

Question 4 : Définition de la classe CNom

Nous allons définir une classe Nom (voir fichier CNom.h) :

Pour simplifier, pour le moment, un nom est constitué d'un seul mot.

Ce mot est une chaîne alphabétique.

Ce Nom est NORMALISER (voir classe CDuree) à la construction en une chaîne en majuscules.

La procédure de lecture doit utiliser la procédure LireString précédente en n'acceptant que les lettres (majuscules ou minuscules). Le nom sera normalisé ensuite.

Les opérateurs de relation sont surchargés comme d'habitude.

La surcharge de l'injecteur est réalisée avec une fonction 'friend' (non membre de la classe) qui renvoie le flux modifié. La définition de cette fonction est très simple :

```
ostream & operator << (ostream & flux, const CNom & Nom)  
{ return flux << Nom.m_Nom ; }
```

La fonction de validation (non membre) EstStrNom contrôle si la chaîne passée en paramètre est alphabétique.

3) Classe CPrenom (Listing à fournir : CPrenom.cxx) :

Question 5 : Définition de la classe CPrenom

Nous allons définir une classe Prenom (voir fichier CPrenom.h) :

La classe CPrenom a la même structure que la classe CNom sauf que le prénom doit être en minuscules.

4) Classe CIdent (Listing à fournir : CIdent.cxx) :

Question 6 : Définition de la classe CIdent

Nous allons définir la classe CIdent (voir fichier CIdent.h)

La classe CIdent est constituée d'un nom et d'un prénom plus un ACRONYME qui sera calculé à la construction.

On peut utiliser plusieurs images de l'identité.

Les relations d'ordre concernent les acronymes.

5) Classe CCpt (Listing à fournir : CCpt.h CCpt.cxx) :

Question 7 : Déclaration et définition de la classe CCpt

Il nous faut définir maintenant une classe compteur qui sera utilisée plus loin.

On vous demande de concevoir entièrement cette classe afin qu'elle puisse répondre le plus simplement possible aux besoins suivants :

- Une donnée membre m_Compteur qui puisse être incrémenté ou décrémenté.
- Une décrémentation sur un compteur nul sera sans effet.

6) Classe CPerso (Listing à fournir : CPerso.cxx) :

Question 8 : Définition de la classe CPerso

Nous allons définir la classe CPerso (voir fichier CPerso.h)

La classe CPerso dérive de la classe CIdent qui est la classe mère de toute les classes qui ont besoin de gérer une identité. En plus de l'identité, la classe CPerso est constituée d'un compteur (qui pourrait représenter une note, par exemple) et d'un classement fonction du compteur.

Les modifieurs permettent d'incrémenter le compteur et de changer le classement (voir plus loin).

La relation d'ordre se fait par rapport au classement. Vous pouvez ajouter les relations absentes.

La surcharge de l'injecteur fournit les données dans une présentation agréable.
La fonction non membre StringToPerso extrait les différentes données de la chaîne Str. Elle pourra être utilisée par l'injecteur.

C. Deuxième Partie : Le Fichier :

Il s'agit maintenant de gérer un fichier de personnes en utilisant les classes ci-dessus.

L'ensemble des personnes est mémorisé dans le fichier texte PersoIn à raison d'une ligne par personne. A l'intérieur de la ligne, les données sont séparées par au moins un espace (on pourra utiliser l'extracteur). On trouvera successivement : Nom, Prénom, Compteur, Classement. Les lignes sont triées dans l'ordre alphabétique des noms puis des prénoms (en fait, selon les acronymes).

1) Mise à jour du classement (Listing à fournir : Acro1.cxx) :

Ce programme consiste à saisir l'acronyme d'une personne, à incrémenter son compteur et à visualiser son classement. Bien entendu, à chaque modification du compteur, le classement doit être mis à jour.

Pour bien comprendre ce que vous devez réaliser, je vous propose d'exécuter mon programme Acronyme1.

On vous demande de réaliser les opérations dans l'ordre où elles sont présentées.

Question 8 : Mémorisation en mémoire dynamique

Lire le fichier d'entrée PersoIn ligne par ligne et le stocker en mémoire dynamique tout en créant un vecteur de pointeurs comme cela a été réalisé en TP. Ce premier vecteur permet donc d'accéder aux personnes dans l'ordre alphabétique puisque le fichier est trié sur les acronymes.

Afficher les personnes dans l'ordre alphabétique pour contrôler.

Question 9 : Recherche dichotomique

Faire une boucle de saisie sur l'acronyme. Rechercher la personne en faisant une recherche dichotomique dans le vecteur de pointeurs. La version de la recherche dichotomique étudiée en cours devra être modifiée pour tenir compte des pointeurs.

En cas de succès, afficher la personne, en cas d'échec, afficher un message d'erreur.

Question 10 : Vector de classement

Ajouter un deuxième vector de pointeurs qui permet d'accéder aux personnes dans l'ordre du classement. Il y a donc deux vectors de pointeurs pour accéder aux données qui elles ne sont mémorisées qu'une seule fois.

Afficher les personnes dans l'ordre du classement pour contrôler.

Question 11 : Mise à jour

Il s'agit maintenant de modifier le compteur en mettant à jour le classement.

Revenir à la question 9 et incrémenter le compteur de la personne lorsque celle-ci a été trouvée. Mettre à jour alors son classement et visualiser les données de la personne modifiée. Attention, cette opération nécessite la mise à jour du vector de classement pour pouvoir accéder aux personnes dans l'ordre du classement.

Question 12 : Accès au classement

Faire une boucle de lecture des classements. Contrôler la validité et afficher la personne correspondante.

Question 13 : Menu

Regrouper les opérations précédentes dans un menu qui permettra de :

- Visualiser une personne d'un acronyme donné.
- Incrémenter son compteur.
- Visualiser une personne d'un classement donné.

Question 14 : Sauvegarde

Ajouter un fichier Journal qui mémorise l'ensemble des opérations réalisées comme on l'a fait en TD pour l'application éditeur.

Sauvegarder les données dans l'ordre alphabétique dans le fichier PersoOut.

2) Ajout de personnes (Listing à fournir : Acro2.cxx) :

Question 15 : Ajout de personnes

Faire une boucle de lecture qui saisit le nom et le prénom d'une nouvelle personne.

Par recherche dichotomique, vérifier son acronyme.

Si l'acronyme existe déjà, refuser l'ajout.

Insérer la nouvelle personne en respectant l'ordre alphabétique.

Bien entendu, le compteur des nouvelles personnes est nul.

Faire attention au classement.

Faire un journal et sauvegarder.

Exécuter de nouveau le programme Acro1 pour contrôler.

3) Suppression de personnes (Listing à fournir : Acro3.cxx) :

Question 16 : Suppression de personnes

Saisir le nom des personnes à supprimer et mettre à jour.

Pensez à mettre à jour le classement avant la sauvegarde.

D. Troisième partie : Compléments

Cette partie ne pourra être envisagée que lorsque tout ce qui précède sera terminé. Elle est facultative et ne sera comptée qu'en bonus (3 points au maximum) .

A la différence des questions précédentes, l'énoncé est assez libre.

Pour rendre ce programme exploitable, vous voyez bien qu'il faut compléter les classes.

Vous pouvez donc ajouter (dans l'ordre d'importance) :

- Traiter les exceptions.
- Compléter la classe CNom en gérant des noms composés.
- Compléter la classe CPrenom en gérant plusieurs prénoms (3 au maximum).
- Ajouter une classe CAdresse.
- Autres améliorations à votre convenance.

*II. Explications concernant
le projet*

I. Présentation du sujet :

L'objectif de cette seconde partie est de réaliser une gestion de stocks. Pour ce faire, nous devons utiliser les deux classes développées en première partie, à savoir les classes `CvecIntGen` et `CvecIntGenMax`.

L'option choisie ici pour la réalisation desdits programmes consiste à séparer l'introduction de nouveaux produits de la mise à jour du stock, respectivement grâce au programme `stock1` et au programme `stock2`.

Enfin, il est à noter le développement du packaging utilitaire d'entrée-sortie, qui permet en particulier la saisie contrôlée d'un `unsigned`.

II. Les choix retenus dans les solutions :

A. `IoUtil.cxx` :

Le premier élément à mettre en place dans ce fichier était le booléen `bool EstStrUnsigned (const std::string & Str)`, dont le but est de renvoyer vrai si le `Str` est l'image d'un `unsigned` précédé et/ou suivi de blanc. Tout d'abord, on recherche dans le `string` via la fonction `find_first_not_of()` si les caractères entrés sont conformes à la demande. Si c'est le cas, il s'agit alors de s'assurer que la valeur entrée n'est pas vide.

Ensuite, fût mis en place la fonction `Purger()`, qui a pour but de se positionner sur `cin` au delà de la première balise `'\n'`.

Puis fût mis en place la fonction `GetUnsigned (const std::string & Invite, unsigned & Val)`, qui a pour but de permettre la saisie d'un `unsigned` avec la répétition de l'invite de commande. A noter l'utilisation de l'algorithme de Hörner dans la fonction.

Enfin, fût mis en place la fonction `Oui (const std::string & Question)`, qui permet de répondre à une éventuelle question par (O/N).

B. `stock1.cxx` :

Notons tout d'abord la création, avant toute chose, dans le namespace, la définition de vecteur de `CString` et de vecteur de `CVecteurMax`, initialisé à la valeur 5, afin de créer respectivement les vecteurs `Reference` et `Quantites`.

Passons maintenant à la fonction `ControlChaine()`, dont le but est de réaliser un contrôle interne sur la chaîne entrée en invite de commande.

Dans un premier temps, nous créons une `string` `Chaine`, créée grâce à `CString`. Ensuite, dans une boucle de saisie, nous réalisons dans un premier un passage de tous les caractères de la chaîne en majuscules grâce à la fonction `ToUpper()` de `CString`.

Vient ensuite les boucles de test. La première, via la fonction `find_first_not_of()` va chercher si un caractère est invalide ou non. Si le caractère est invalide, une exception est renvoyée. La seconde permet de vérifier dans un premier temps que le premier caractère de la chaîne est bien une lettre, via la fonction `isalpha()`, et que la chaîne fait bien 5 caractères, via la fonction `size()`. Dans un second temps, il s'agit de vérifier que si un chiffre apparaît dans la suite de caractères, il n'est suivi que de chiffres, ce qui est vérifié par l'utilisation successive des fonction `find_first_not_of()` et `find_first_of()`. Comme précédemment, si une de ces conditions n'est pas remplie, une exception est renvoyée. La troisième boucle a pour but de vérifier que la référence entrée n'est pas déjà existante, auquel cas une exception est renvoyée.

Enfin, une boucle de confirmation utilisant la fonction `Oui()` de `IoUtil.cxx` est mis en place.

Passons maintenant au contenu à proprement parlé du `main()`.

Notons tout d'abord la présence du code de remise à zéro de l'écran en début de `main`, code vu dans le `Tp1` de programmation.

Ensuite, la lecture du fichier `StockIn`, fichier où les données de départ sont enregistrées, est mis en place. Ainsi, un flux d'entrée est créé, ainsi qu'une `string` temporaire `Temp`. La lecture du `string` est subdivisée en deux parties `Nom` et `Vec` (`Nom` pour la référence, et `Vec` pour les valeurs correspondant à la référence). Pour ce faire, on utilise la fonction `substr()`. Le résultat est ensuite affiché à l'écran.

On trouve par la suite la présence d'un bloc `try-catch`, qui permet de capturer l'exception lorsqu'elle se présente et d'afficher son code. (Ici, il a été choisi d'afficher un message, par soucis de clarté pour l'utilisateur).

Est ensuite créé un flux de sortie, afin de créer le fichier de sortie `StockOut`. Chaque nouvelle référence y est insérée, avec une mise à zéro de tous ses éléments.

On notera enfin la présence de la fonction `sleep()`. Dans un souci de clarté, le code de remise à zéro de l'écran a aussi été inséré en fin de programme. Le problème qui s'est alors présenté est que le calcul se faisant très rapidement, l'affichage de la confirmation d'écriture du fichier `StockOut` paraissait comme « invisible ». L'idée qui est alors apparue était alors de ralentir un tout petit peu l'exécution de cet affichage, afin que l'utilisateur ai la confirmation de l'écriture du fichier `StockOut`, ce qui a été réalisé avec la fonction `sleep()`, dont l'include est `<unistd.h>` et qui prends en paramètres des entiers `int` en seconde.

C. `stock2.cxx` :

Notons tout d'abord la création, avant toute chose, dans le namespace, la définition de vecteur de `CString` et de vecteur de `CVecteurMax`, initialisé à la valeur 5, afin de créer respectivement les vecteurs `Reference` et `Quantites`.

Passons maintenant à la fonction `ControlChaine()`, dont le but est de réaliser un contrôle interne sur la chaîne entrée en invite de commande.

Dans un premier temps, nous créons une string `Chaine`, créée grâce à `CString`. Ensuite, dans une boucle de saisie, nous réalisons dans un premier un passage de tous les caractères de la chaîne en majuscules grâce à la fonction `ToUpper()` de `CString`.

Vient ensuite les boucles de test. La première, via la fonction `find_first_not_of()` va chercher si un caractère est invalide ou non. Si le caractère est invalide, une exception est renvoyée. La seconde permet de vérifier dans un premier temps que le premier caractère de la chaîne est bien une lettre, via la fonction `isalpha()`, et que la chaîne fait bien 5 caractères, via la fonction `size()`. Dans un second temps, il s'agit de vérifier que si un chiffre apparaît dans la suite de caractères, il n'est suivi que de chiffres, ce qui est vérifié par l'utilisation successive des fonction `find_first_not_of()` et `find_first_of()`. Comme précédemment, si une de ces conditions n'est pas remplie, une exception est renvoyée. La troisième boucle a pour but de vérifier que la référence entrée n'est pas déjà existante. auquel cas une exception est renvoyée.

On a ensuite la création d'un vecteur `Vec` de `CVecteurMax`, qui peut contenir 5 éléments et d'un `unsigned` nommé `Nb`. Grâce à la fonction `SetElem()`, on va changer la valeur de chacun des 5 éléments en lui donnant une valeur `Nb`, valeur que l'utilisateur entre au clavier grâce à la fonction `GetUnsigned()`.

Enfin, une boucle de confirmation utilisant la fonction `Oui()` de `IoUtil.cxx` est mise en place.

A noter enfin la présence de la fonction `Tri()`, qui fonctionne selon le modèle du `TriBulle`. (La fonction a été vu plusieurs fois en amphi sous le nom `BubbleSort()`). Cette fonction a pour but de déterminer l'élément le plus grand du vecteur.

Passons maintenant au contenu à proprement parlé du `main()`.

Notons tout d'abord la présence du code de remise à zéro de l'écran en début de `main`, code vu dans le `Tp1` de programmation.

Ensuite, la lecture du fichier `StockIn`, fichier où les données de départ sont enregistrées, est mis en place. Ainsi, un flux d'entrée est créé, ainsi qu'une string temporaire `Temp`. La lecture du string est subdivisée en deux parties `Nom` et `Vec` (nom pour la référence, et `Vec` pour les valeurs correspondant à la référence). Pour ce faire, on utilise la fonction `substr()`. Le résultat est ensuite affiché à l'écran.

On trouve par la suite la présence d'un bloc `try-catch`, qui permet de capturer l'exception lorsqu'elle se présente et d'afficher son code. (Ici, il a été choisi d'afficher un message, par soucis de clarté pour l'utilisateur).

Est ensuite créé un flux de sortie, afin de créer le fichier de sortie `StockOut`. Chaque nouvelle référence `y` est insérée, avec une mise à zéro de tous ses éléments.

On notera enfin la présence de la fonction `sleep()`. Dans un soucis de clarté, le code de remise à zéro de l'écran a aussi été inséré en fin de programme. Le problème qui s'est alors présenté est que le calcul se faisant très rapidement, l'affichage de la confirmation d'écriture du fichier `StockOut` paraissait comme « invisible ». L'idée qui est alors apparue était alors de ralentir un tout petit peu l'exécution de cet affichage, afin que l'utilisateur ai la confirmation de l'écriture du fichier `StockOut`, ce qui a été réalisé avec la fonction `sleep()`, dont l'include est `<unistd.h>` et qui prends en paramètres des entiers `int` en seconde.

III. CString.h


```

////////////////////
//          //
// Controles //
//          //
////////////////////

bool EstChaineVide () const;
bool EstChaineEspace () const;
bool EstChaineMajuscule () const;
bool EstChaineMinuscule () const;
bool EstChaineMinusculeAccent () const;
bool EstChaineSansAccent () const;
bool EstChaineAlpha () const;
bool EstChaineAlphaAccent () const;
bool EstChaineAlphaAccentEspace () const;
bool EstChaineNum () const;
bool EstChaineHexa () const;
bool EstChaineAlphaNum () const;
bool EstChaineAlphaNumAccent () const;
bool EstChaineAlphaNumAccentEspace () const;
bool EstChaineEditable () const;
bool EstChaineGraphique () const;
bool EstChaineBinaire () const;
bool EstChaineMono () const;
bool EstChaineAvecCaracteresAutorises
    (const char* const CaracteresAutorises) const;
bool EstChaineSansCaracteresInterdits
    (const char *const CaracteresInterdits) const;
bool EstCaracPresent (const char c) const;

////////////////////
//          //
// Edition  //
//          //
////////////////////

CString Copier (const Ind_t Deb = 0,
                const Ind_t Nbre = npos) const;
CString Move (const Ind_t Taille,
              const Sens_t Cadrage = CstGauche,
              const char Complement = ' ') const;

void Couper (const Ind_t Deb = 0,
             const Ind_t Nbre = npos);

void Coller (const CString & Elem, const Ind_t Deb);
void Coller (const char* Elem, const Ind_t Deb);
void Coller (const char Elem, const Ind_t Deb);

```

```

void CollderDebut (const CString & Elem);
void CollderDebut (const char* Elem);
void CollderDebut (const char Elem);

void CollderFin (const CString & Elem);
void CollderFin (const char* Elem);
void CollderFin (const char Elem);

void Changer (const char* const Ancien,
              const char* const Nouveau);
void Changer (const char Ancien, const char Nouveau);

////////////////////
//                //
// Transformations //
//                //
////////////////////

void ToMajuscule ();
void ToMinuscule ();
void ToSansAccent ();
void ToEditable (const char c = ' ');
void UnsignedToString (void);
void Inverser ();
void Cadrer (const Sens_t Cadrage = CstGauche);

////////////////////
//                //
// Justifications //
//                //
////////////////////

void Compacter();
void Normaliser();
void JustifierBords (const Sens_t Cadrage = CstGauche);
void Justifier();

```

```

//////////
//          //
// Comptages //
//          //
//////////

int NbreMots() const;
int NbreOccurrences (const char c) const;
int NbreOccurrences (const char* const Str) const;
int NbreOccurrences (const CString & Str) const;
int NbreDevant (const char c = ' ') const;
int NbreDerriere (const char c = ' ') const;
int TailleUtile() const;

//////////
//          //
// Lecture //
//          //
//////////

typedef bool (* FctValidChar_t) (const char);

void LireString (FctValidChar_t isGood,
                const bool AvecEcho = true);

}; // CString

std::string UnsignedToString (const unsigned i);
int CharHexToInt (char c);
unsigned StringToUnsigned (const std::string & Str);

} // nsMonSpace

#endif

```

IV. CString.cxx

```

/**
 *
 * @ File : CString.cxx
 *
 * @ Authors : V.Castille, M.Lapeyre
 *
 * @ Date : 06/03/05
 *
**/

#if !defined __CSTRING_CXX__
#define __CSTRING_CXX__

#include <iostream>
#include <cctype>

#include "CString.h"
#include "Char.h"
#include "nsUtil.h"

using namespace std;
using namespace nsUtil;
using namespace nsMonSpace;

#define CSTR nsMonSpace::CString

//////////
//          //
// Constructeurs //
//          //
//////////

CSTR::CString(const char* const Str)
    :string (Str) {}

CSTR::CString(const std::string & Str)
    :string (Str) {}

CSTR::CString(const Ind_t Taille, const char c)
    :string (Taille, c) {}

```

```

CSTR::CString(const Ind_t Taille, const char* const Str)
    :string (Taille, ' ')
{
    for (Ind_t i (0), j (0); i < size (); ++i)
    {
        at (i) = Str[j++];

        if ('\0' == Str[j])
            j = 0;

    } // for()
}

CSTR::CString(const CString & Origine, const Ind_t Deb, const Ind_t
Nbre)
{
    if (Deb >= Origine.size())
        throw out_of_range("out_of_range / CString::CString");

    Ind_t Last = (Nbre == npos) ? Origine.size() : Deb + Nbre;

    if ( Last > Origine.size() )
        Last = Origine.size();

    resize (Last - Deb);

    Ind_t i (0);

    for (Ind_t k (Deb); k < Last; ++k)
        at(i++) = Origine[k];

} // Copier

```

```

CSTR::CString(const Ind_t Taille, const CString & Origine,
              const Sens_t Cadrage, const char Complement)
: string (Taille, Complement)
{
    unsigned Cpt = ( Taille < Origine.size() )
                  ? Taille : Origine.size();

    if (Cadrage == CstGauche)
        for (Ind_t i (0); Cpt; ++i, --Cpt)
            at (i) = Origine[i];
    else
        for (Ind_t i (Taille -1), j (Origine.size() - 1);
             Cpt; --i, --j, --Cpt)
            at (i) = Origine[j];

} // Move

//////////
//          //
// Contrôles //
//          //
//////////

bool CSTR::EstChaineVide() const
{
    return (0 == size());
} // EstChaineVide()

bool CSTR::EstChaineEspace() const
{
    for (Ind_t i (0); i < size(); ++i)
        if (!nsMonSpace::EstCharEspace( at(i) ))
            return false;

    return true;
} // EstChaineEspace()

bool CSTR::EstChaineMajuscule() const
{
    for (Ind_t i (0); i < size(); ++i)
        if (!nsMonSpace::EstCharMajuscule( at(i) ))
            return false;

    return true;
} // EstChaineMajuscule()

```

```

bool CSTR::EstChaineMinuscule() const
{
    for (Ind_t i (0); i < size(); ++i)
        if (!nsMonSpace::EstCharMinuscule( at (i) ))
            return false;

    return true;
} // EstChaineMinuscule()

bool CSTR::EstChaineMinusculeAccent() const
{
    for (Ind_t i (0); i < size(); ++i)
        if (!nsMonSpace::EstCharMinusculeAccent( at(i) ))
            return false;

    return true;
} // EstChaineMinusculeAccent()

bool CSTR::EstChaineSansAccent() const
{
    for (Ind_t i (0); i < size(); ++i)
        if (nsMonSpace::EstCharAccentue( at(i) ))
            return false;

    return true;
} // EstChaineSansAccent()

bool CSTR::EstChaineAlpha() const
{
    for (Ind_t i (0); i < size(); ++i)
        if (!nsMonSpace::EstCharAlpha( at(i) ))
            return false;

    return true;
} //EstChaineAlpha()

```

```

bool CSTR::EstChaineAlphaAccent() const
{
    for (Ind_t i (0); i < size(); ++i)
        if (!nsMonSpace::EstCharAlphaAccent( at(i) ))
            return false;

    return true;
} // EstchaineAlphaAccent()

bool CSTR::EstChaineAlphaAccentEspace() const
{
    for (Ind_t i (0); i < size(); ++i)
        if (!nsMonSpace::EstCharAlphaAccentEspace( at(i) ))
            return false;

    return true;
} // EstChaineAlphaAccentEspace()

bool CSTR::EstChaineNum() const
{
    for (Ind_t i (0); i < size(); ++i)
        if (!nsMonSpace::EstCharNum( at(i) ))
            return false;

    return true;
} // EstChaineNum()

bool CSTR::EstChaineHexa() const
{
    for (Ind_t i (0); i < size(); ++i)
        if (!nsMonSpace::EstCharHexa( at(i) ))
            return false;

    return true;
} // EstChaineHexa()

```

```

bool CSTR::EstChaineAlphaNum() const
{
    for (Ind_t i (0); i < size(); ++i)
        if (!nsMonSpace::EstCharAlphaNum( at(i) ))
            return false;

    return true;
} // EstChaineAlphaNum()

bool CSTR::EstChaineAlphaNumAccent() const
{
    for (Ind_t i (0); i < size(); ++i)
        if (!nsMonSpace::EstCharAlphaNumAccent( at(i) ))
            return false;

    return true;
} // EestChaineAlphaNumAccent()

bool CSTR::EstChaineAlphaNumAccentEspace() const
{
    for (Ind_t i (0); i < size(); ++i)
        if (!nsMonSpace::EstCharAlphaNumAccentEspace( at(i) ))
            return false;

    return true;
} // EstChaineAlphaNumAccentEspace()

bool CSTR::EstChaineEditable() const
{
    for (Ind_t i (0); i < size(); ++i)
        if (!nsMonSpace::EstCharEditable( at(i) ))
            return false;

    return true;
} // EstChaineEditable()

```

```

bool CSTR::EstChaineGraphique() const
{
    for (Ind_t i (0); i < size(); ++i)
        if (!nsMonSpace::EstCharGraphique( at(i) ))
            return false;

    return true;
} // EstChaineGraphique()

bool CSTR::EstChaineBinaire() const
{
    for (Ind_t i (0); i < size(); ++i)
        if (!nsMonSpace::EstCharBinaire( at(i) ))
            return false;

    return true;
} // EstChaineBinaire()

bool CSTR::EstChaineMono() const
{
    for (Ind_t i (0); i < size(); ++i)
        if (at(i) != at(0))
            return false;

    return true;
} // EstChaineMono()

bool CSTR::EstChaineAvecCaracteresAutorises
                                     (const char* CaracteresAutorises) const
{
    for (Ind_t i (0); i < size(); ++i)
    {
        bool Trouve = false;

        for (Ind_t j (0); CaracteresAutorises[j] != '\0'; ++j)
            if (at(i) == CaracteresAutorises[j])
                Trouve = true;

        if (!Trouve) return false;
    } // for()

    return true;
} // EstChaineAvecCaracteresAutorises()

```

```

bool CSTR::EstChaineSansCaracteresInterdits
                                     (const char *CaracteresInterdits) const
{
    for (Ind_t i (0); i < size(); ++i)
    {
        bool Trouve = false;

        for (Ind_t j (0); CaracteresInterdits[j] != '\0'; ++j)
            if (at(i) == CaracteresInterdits[j])
                Trouve = true;

        if (Trouve) return false;
    } // for()

    return true;
} // EstChaineSansCaracteresInterdits()

bool CSTR::EstCaracPresent (const char c) const
{
    return (0 != NbreOccurences(c));
} // EstCaracPresent()

//////////
//      //
// Edition //
//      //
//////////

CSTR CSTR::Copier (const Ind_t Deb, const Ind_t Nbre) const
{
    return CString(*this, Deb, Nbre);
} // Copier

CSTR CSTR::Move (const Ind_t Taille, const Sens_t Cadrage,
                 const char Complement) const
{
    CString Local(Taille, Complement);

    if (Cadrage == CstGauche)
        for (unsigned i (0); i < Taille && i < size (); ++i)
            Local [i] = (*this) [i];
}

```

```

        else
        {
            unsigned i (Taille - 1), j (size () - 1);

            for ( ; i > 0 && j > 0; --i, --j)
                Local [i] = (*this) [j];

        } // else

        return Local;

    } // Move()

void CSTR::Couper (const Ind_t Deb, const Ind_t Nbre)
{
    if (Deb >= size())
        throw out_of_range("out_of_range / CString::Couper");

    erase (Deb, Nbre);

} // Couper

void CSTR::Coller (const CString & Elem, const Ind_t Deb)
{
    if (Deb > size())
        throw out_of_range("out_of_range / CString::Coller");

    insert (Deb, Elem);

} // Coller

void CSTR::Coller (const char* Elem, const Ind_t Deb)
{
    if (Deb > size())
        throw out_of_range("out_of_range / CString::Coller");

    insert (Deb, Elem);

} // Coller

void CSTR::Coller (const char Elem, const Ind_t Deb)
{
    if (Deb > size())
        throw out_of_range("out_of_range / CString::Coller");

    insert (Deb, string(1, Elem));

} // Coller

```

```
void CSTR::CollerDebut (const CString & Elem)
{
    Coller (Elem, 0);
} // Coller

void CSTR::CollerDebut (const char *Elem)
{
    Coller (Elem, 0);
} // Coller

void CSTR::CollerDebut (const char Elem)
{
    Coller (Elem, 0);
} // Coller

void CSTR::CollerFin (const CString & Elem)
{
    Coller (Elem, size());
} // Coller

void CSTR::CollerFin (const char* Elem)
{
    Coller (Elem, size());
} // Coller

void CSTR::CollerFin (const char Elem)
{
    Coller (Elem, size());
} // Coller
```

```

void CSTR::Changer (const char* const Ancien,
                   const char* const Nouveau)
{
    CString StrAncien (Ancien);

    Ind_t Long (StrAncien.size());

    for (Ind_t i (0); i <= (size() - Long); ++i)
    {
        if (Copier (i, Long) == StrAncien)
        {
            Couper (i, Long);
            Coller (Nouveau, i);
        } // if()
    } // for()

} // Changer

void CSTR::Changer (const char Ancien, const char Nouveau)
{
    for (Ind_t i (0); i < size(); ++i)
        if (at(i) == Ancien)
            at(i) = Nouveau;

} // Changer

////////////////////////////////////
//                               //
// Transformations //
//                               //
////////////////////////////////////

void CSTR::ToMajuscule()
{
    for (Ind_t i (0); i < size(); ++i)
        at(i) = nsMonSpace::ToMajuscule( at(i) );

} // ToMajuscule

void CSTR::ToMinuscule()
{
    for (Ind_t i (0); i < size(); ++i)
        at(i) = nsMonSpace::ToMinuscule( at(i) );

} // ToMinuscule

```

```

void CSTR::ToSansAccent()
{
    for (Ind_t i (0); i < size(); ++i)
        at(i) = nsMonSpace::ToSansAccent( at(i) );
} // ToSansAccent

void CSTR::ToEditable(const char c)
{
    for (Ind_t i (0); i < size(); ++i)
        if (!EstCharEditable (c))
            at(i) = c;
} // ToEditable

void CSTR::Inverser ()
{
    char c;

    for (int i=0, j=size()-1; i < j; ++i, --j)
    {
        c = at(i); at(i)=at(j); at(j)=c;
    } // for()
} // Inverser

void CSTR::Cadrer (const Sens_t Cadrage)
{
    if (Cadrage == CstGauche)
    {
        Ind_t i (0);
        Ind_t j (find_first_not_of (' '));

        if ( (0 == j) || (npos == j) )
            return;

        for (; j < size (); ++j, ++i)
            at(i) = at(j);

        for (; i < size (); ++i)
            at(i) = ' ';

        return;
    } // if()
}

```

```

if (Cadrage == CstDroite)
{
    int i (size()-1);
    int j (find_last_not_of ( ' ' ));

    if ( (size()-1 == j) || (npos == j) )
        return;

    for (; j >= 0; j--)
        at(i--) = at(j);

    for (; i >= 0; i--)
        at(i) = ' ';

    return;

} // if()

if (EstChaineEspace ())
    return;

int i (size()-1);
int j (find_last_not_of ( ' ' ));

for (; j >= 0; j--)
    at(i--) = at(j);

for (; i >= 0; i--)
    at(i) = ' ';

j = find_first_not_of ( ' ' );
i = j / 2;

for (; j < size (); ++j, ++i)
    at(i) = at(j);

for (; i < size (); ++i)
    at(i) = ' ';

} // Cadrer

```

```

void CSTR::Compacter ()
{
    Ind_t j (0);

    for (int i (0); i < size(); ++i)
        if (' ' != at(i))
            at(j++) = at(i);

    resize (j);
} // Compacter

void CSTR::Normaliser ()
{
    Compacter ();
    ToMajuscule ();
} // Normaliser

void CSTR::JustifierBords (const Sens_t Cadrage)
{
    if (EstChaineEspace())
    {
        resize (0);

        return;
    } // if ()

    if (CstGauche == Cadrage)
    {
        int i (find_first_not_of (' '));

        Cadrer (CstGauche);
        resize (size() - i);

        return;
    } // if ()

```

```

if (CstDroite == Cadrage)
{
    int i (find_last_not_of ( ' ' ));

    resize (i + 1);
    return;

} // if ()

Cadrer (CstGauche);

int i (find_last_not_of ( ' ' ));

resize (i + 1);

} // JustifierBords

void CSTR::Justifier()
{
    Ind_t NewSize = 0;

    if (!EstChaineEspace())
    {
        char Precedent = ' ';

        Ind_t i;

        for (i = 0; i < size(); ++i)
        {
            if ( !(EstCharEspace (at(i))
                && EstCharEspace (Precedent)) )
                at(NewSize++) = at(i);

            Precedent = at(i);

        } // for()

        if (EstCharEspace( at(--i) )) NewSize--;

    } // if()

    resize(NewSize);

} // Justifier

```

```

int CSTR::NbresMots() const
{
    char Precedent = ' ';

    int Cpt (0);

    for (int i (0); i < size(); ++i)
    {
        if ( !EstCharEspace(at(i)) && EstCharEspace(Precedent) )
            Cpt++;

        Precedent = at(i);
    } // for()

    return Cpt;
} // NbresMots

int CSTR::NbresOccurences (const char c) const
{
    int Cpt (0);

    for (int i (0); i < size(); ++i)
        if ( at(i) == c )
            Cpt++;

    return Cpt;
} // NbresOccurences

int CSTR::NbresOccurences (const char* const Str) const
{
    if (size() < strlen(Str)) return 0;

    int Cpt (0);

    for (int i (0); i <= size() - strlen(Str); ++i)
    {
        Cpt++;
    }
}

```

```

        for (Ind_t k = i, j = 0; Str[j] != '\0'; ++k, ++j)
            if (at(k) != Str[j])
            {
                Cpt--;
                break;
            } // if()

    } // for()

    return Cpt;

} // NbreOccurences

int CSTR::NbreOccurences (const CString & Str) const
{
    return NbreOccurences(Str.c_str());
} // NbreOccurences

int CSTR::NbreDevant (const char c) const
{
    int i (find_first_not_of (c));

    if (npos == i)
        return size();

    return i;
} // NbreDevant

int CSTR::NbreDerriere (const char c) const
{
    int i (find_last_not_of (c));

    if (npos == i)
        return size();

    return size() - i - 1;
} // NbreDerriere

int CSTR::TailleUtile () const
{
    return size() - NbreDerriere();
} // TailleUtile

```

```

void CSTR::LireString (FctValidChar_t isGood, const bool AvecEcho)
{
    const unsigned char CstBackSpace = 0x8;
    const unsigned char CstBell = 0x7;
    const unsigned CstDel = 0x7f;
    const unsigned CstCr = 13;

    clear();
    char c;
    KBdDirectOn ();

    cout << flush; // Vidage tampon

    for ( ; ; )
    {
        c = KBdDirectRead ();

        if (CstCr == c) break;

        if (isGood (c))
        {
            push_back (c);

            if (!AvecEcho) c = '*';

            cout << c << flush;

        } // if ()
        else
            if (CstDel == c && 0 != size ())
            {
                erase (end () - 1);

                cout << CstBackSpace
                    << ' ' << CstBackSpace << flush;

            } // if()
        else
            cout << CstBell << flush;

    } // for

    KBdDirectOff ();

} // LireString

```

```

////////////////////////////////////
//                               //
// Fonctions non membres //
//                               //
////////////////////////////////////

string nsMonSpace::UnsignedToString (const unsigned i)
{
    CString Result;

    if (0 == i)
        Result.push_back ('0');
    else
        for (unsigned Nb = i; Nb > 0; Nb /= 10)
            Result.push_back (ToChar (Nb % 10));

    Result.Inverser ();

    return Result;
} // UnsignedToString()

int nsMonSpace::CharHexToInt (char c)
{
    return isxdigit (c)
        ? isdigit (c)
            ? c - '0'
            : toupper (c) - 'A' + 10
        : -1;
} // CharHexToInt()

unsigned nsMonSpace::StringToUnsigned (const std::string & Str)
{
    unsigned Res = 0;
    const unsigned NbDigit = Str.size();

    for (string::size_type i (0) ; i < NbDigit ; )
        Res = Res * 10 + CharHexToInt (Str [i++]);

    return Res;
} // StringToUnsigned ()

#undef CSTR

#endif

```

V. CNom.h

```

// Gestion du type Nom
// Fichier CNom.h
// Ecrit par P. SEBILO le 21.1.2005
// Dernière mise à jour le 1.2.2005
// Modifié le 22/03/05 par V.Castille et M.Lapeyre

#if !defined __CNOM_H__
#define __CNOM_H__

#include "CString.h"
#include "Char.h"

namespace nsMonSpace
{
    class CNom
    {
        protected :

            CString m_Nom;
            void Normaliser (void) throw ();

        public :

            CNom (const std::string & Nom) throw ();

            std::string GetNom (void) const throw ();

            std::string GetDebutNom
                (const unsigned NbLettres) const throw ();

            void Lire (void) throw ();

            bool operator < (const CNom & Nom) throw ();
            bool operator > (const CNom & Nom) throw ();
            bool operator <= (const CNom & Nom) throw ();
            bool operator >= (const CNom & Nom) throw ();
            bool operator == (const CNom & Nom) throw ();
            bool operator != (const CNom & Nom) throw ();

            friend std::ostream & operator <<
                (std::ostream & flux, const CNom & Nom);
    };

    bool EstStrNom (const CString & Str) throw ();
} // nsMonSpace

#endif

```

VI. CNom.cxx

```

/**
 *
 * @ File : CNom.cxx
 *
 * @ Authors : V.Castille, M.Lapeyre
 *
 * @ Date : 08/03/04
 *
**/

#if !defined __CNOM_CXX__
#define __CNOM_CXX__

#include "CNom.h"

#include <cctype>

using namespace std;

#define SSTR std::string

namespace nsMonSpace
{
    //////////////////////////////////
    //                //
    // Traitement //
    //                //
    //////////////////////////////////

    void CNom::Normaliser (void) throw()
    {
        m_Nom.ToMajuscule();
    } // Normaliser ()

```

```

////////////////////////////////////
//                               //
// Constructeur //
//                               //
////////////////////////////////////

CNom::CNom (const SSTR & Nom) throw ()
    : m_Nom (Nom)
{
    Normaliser();
}

////////////////////////////////////
//                               //
// Accesseurs //
//                               //
////////////////////////////////////

SSTR CNom::GetNom (void) const throw()
{
    return m_Nom;
} // GetNom ()

SSTR CNom::GetDebutNom (unsigned NbLettres) const throw()
{
    if (m_Nom.size() < NbLettres)
        NbLettres = m_Nom.size();

    SSTR Deb (NbLettres, ' ');

    for (SSTR::size_type i = 0 ; i < NbLettres ; ++i)
    {
        if (i >= m_Nom.size())
            break;
        Deb[i] = m_Nom[i];
    } // for ()

    return Deb;
} // GetDebutNom ()

```

```

void CNom::Lire (void) throw ()
{
    m_Nom.LireString(EstCharAlpha);

    Normaliser();

} // Lire ()

////////////////////////////////////
//                               //
// Opérateurs (classés selon    //
// la logique du code de Gray) //
//                               //
////////////////////////////////////

bool CNom::operator < (const CNom & Nom) throw ()
{
    return m_Nom < Nom.m_Nom;

} // operator <

bool CNom::operator <= (const CNom & Nom) throw ()
{
    return m_Nom <= Nom.m_Nom;

} // operator <=

bool CNom::operator == (const CNom & Nom) throw ()
{
    return m_Nom == Nom.m_Nom;

} // operator ==

bool CNom::operator != (const CNom & Nom) throw ()
{
    return m_Nom != Nom.m_Nom;

} // operator !=

bool CNom::operator >= (const CNom & Nom) throw ()
{
    return m_Nom >= Nom.m_Nom;

} // operator >=

```

```
bool CNom::operator > (const CNom & Nom) throw ()
{
    return m_Nom > Nom.m_Nom;
} // operator >

ostream & operator << (ostream & flux, const CNom & Nom)
{
    return flux << Nom.m_Nom;
} // operator <<

bool EstStrNom (const CString & Str) throw ()
{
    return Str.EstChaineAlpha();
} // EstStrNom ()

} // namespace nsMonSpace

#undef SSTR

#endif
```

VII. CPrenom.h

```

// Gestion du type Prenom
// Fichier CPrenom.h
// Ecrit par P. SEBILO le 21.1.2005
// Dernière mise à jour le 28.1.2005
// Mise à jour le 22/03/05 par V.Castille et M.Lapeyre

#if !defined __CPRENOM_H__
#define __CPRENOM_H__

#include "CString.h"
#include "Char.h"

namespace nsMonSpace
{
    class CPrenom
    {
        protected :

            CString m_Prenom;
            void Normaliser (void) throw ();

        public :

            CPrenom (const std::string & Prenom = "") throw ();

            std::string GetPrenom (void) const throw ();
            std::string GetDebutPrenom
                (const unsigned NbLettres) const throw ();

            void Lire (void) throw ();

            bool operator < (const CPrenom & Prenom) throw ();
            bool operator > (const CPrenom & Prenom) throw ();
            bool operator <= (const CPrenom & Prenom) throw ();
            bool operator >= (const CPrenom & Prenom) throw ();
            bool operator == (const CPrenom & Prenom) throw ();
            bool operator != (const CPrenom & Prenom) throw ();

            friend std::ostream & operator <<
                (std::ostream & flux, const CPrenom & Prenom);
    };

    bool EstStrPrenom (const CString & Str) throw ();
} // nsMonSpace

#endif

```

VIII. CPrenom.cxx

```

/**
 * @ File : CPrenom.cxx
 *
 * @ Authors : V.Castille, M.Lapeyre
 *
 * @ Date : 09/03/05
 *
 **/

#ifndef __CPRENOM_CXX__
#define __CPRENOM_CXX__

#include <cctype>
#include <iostream>
#include <sstream>

#include "CPrenom.h"
#include "CString.h"

using namespace std;

namespace nsMonSpace
{
    ////////////////
    //              //
    // Traitement //
    //              //
    ////////////////

    void CPrenom::Normaliser (void) throw ()
    {
        if (m_Prenom.size() != 0)
        {
            m_Prenom.ToMinuscule ();
            m_Prenom [0] = toupper (m_Prenom [0]);

        } // if()

    } // Normaliser()

```

```

////////////////////////////////////
//                               //
// Constructeur //
//                               //
////////////////////////////////////

CPrenom::CPrenom (const std::string & Prenom) throw ()
    :m_Prenom (Prenom)
{
    Normaliser();
}

////////////////////////////////////
//                               //
// Accesseur //
//                               //
////////////////////////////////////

string CPrenom::GetPrenom (void) const throw ()
{
    return m_Prenom;
} // GetPrenom ()

string CPrenom::GetDebutPrenom
                                (const unsigned NbLettres) const throw()
{
    string Local (NbLettres, ' ');

    for (string::size_type i (0) ; i < NbLettres ; ++i)
    {
        if (i == m_Prenom.size())
            break;

        Local [i] = m_Prenom [i];
    } // for()

    return Local;
}

```

```

////////////////////////////////////
//                               //
// Contrôleur //
//                               //
////////////////////////////////////

void CPrenom::Lire (void) throw ()
{
    m_Prenom.LireString (EstCharAlpha);
    Normaliser();

} // Lire ()

////////////////////////////////////
//                               //
// Opérateurs (classés selon //
// la logique du code de Gray) //
//                               //
////////////////////////////////////

bool CPrenom::operator < (const CPrenom & Prenom) throw ()
{
    return m_Prenom < Prenom.m_Prenom;

} // operator <

ostream & operator << (ostream & flux, const CPrenom & Prenom)
{
    return flux << Prenom.m_Prenom;

} // operator <<

bool CPrenom::operator <= (const CPrenom & Prenom) throw ()
{
    return m_Prenom <= Prenom.m_Prenom;

} // operator <=

bool CPrenom::operator == (const CPrenom & Prenom) throw ()
{
    return m_Prenom == Prenom.m_Prenom;

} // operator ==

```

```

bool CPrenom::operator != (const CPrenom & Prenom) throw ()
{
    return m_Prenom != Prenom.m_Prenom;
} // operator !=

bool CPrenom::operator >= (const CPrenom & Prenom) throw ()
{
    return m_Prenom >= Prenom.m_Prenom;
} // operator >=

bool CPrenom::operator > (const CPrenom & Prenom) throw ()
{
    return m_Prenom > Prenom.m_Prenom;
} // operator >

//////////
//          //
// Compareurs //
//          //
//////////

bool EstStrPrenom (const CString & Str) throw ()
{
    return Str.EstChaineAlpha ();
} // EstStrPrenom ()

} // namespace nsMonSpace

#endif

```

IX. CIdent.h

```

// Gestion du type Ident (Nom + Prenom)
// Fichier CIdent.h
// Ecrit par P. SEBILO le 27.1.2005
// Dernière mise à jour le 31.1.2005
// Dernière mise à jour le 22.03.2005 par V.Castille et M.Lapeyre

#if !defined __CIDENT_H__
#define __CIDENT_H__

#include "CNom.h"
#include "CPrenom.h"

#include "CString.h"
#include "Char.h"

namespace nsMonSpace
{
    class CIdent
    {
        protected :

            CNom m_Nom;
            CPrenom m_Prenom;

            CString m_Acronyme;

            void Normaliser (void) throw ();

        public :

            //////////////////////////////////////////////////
            //                               //
            // Constructeur //
            //                               //
            //////////////////////////////////////////////////

            CIdent (const CString & NomPrenom = "") throw ();

```

```

////////////////////////////////////
//                               //
// Accesseurs //
//                               //
////////////////////////////////////

std::string GetAcronyme (void)      throw ();
std::string GetNom      (void) const throw ();
std::string GetPrenom  (void) const throw ();

std::string NomPrenom  (void) const throw ();

std::string PrenomNom  (void) const throw ();

std::string PNom      (void) const throw ();

////////////////////////////////////
//                               //
// Comparaison des acronymes //
//                               //
////////////////////////////////////

bool operator < (const CIdent & Ident) throw ();
bool operator > (const CIdent & Ident) throw ();
bool operator <= (const CIdent & Ident) throw ();
bool operator >= (const CIdent & Ident) throw ();
bool operator == (const CIdent & Ident) throw ();
bool operator != (const CIdent & Ident) throw ();

friend std::ostream & operator <<
    (std::ostream & flux, const CIdent & Ident);

}; // Fermeture de classe

bool EstStrIdent (const CString & Str) throw ();

} // nsMonSpace

#endif

```

X. CIdent.cxx

```

/**
 *
 * @ File : CIdent.cxx
 *
 * @ Authors : V.Castille, M.Lapeyre
 *
 * @ Date : 10/03/04
 *
**/

#if !defined __CIDENT_CXX__
#define __CIDENT_CXX__

#include <sstream>

#include "CString.h"
#include "CNom.h"
#include "CPrenom.h"
#include "Char.h"
#include "CIdent.h"

using namespace std;

namespace nsMonSpace
{
    //////////////////////////////////
    //                          //
    // Traitement //
    //                          //
    //////////////////////////////////

    void CIdent::Normaliser (void) throw ()
    {
        m_Acronyme = m_Nom.GetDebutNom (3) +
                    m_Prenom.GetDebutPrenom (3);
        m_Acronyme.ToMajuscule ();
    } // Normaliser ()

```

```

CIdent::CIdent (const CString & NomPrenom) throw ()
{
    unsigned Nb = NomPrenom.NbreMots ();

    if (Nb != 0)
    {
        istringstream istr (NomPrenom);
        string Nom;

        istr >> Nom;

        m_Nom = CNom (Nom);

        if (Nb != 1)
        {
            string Prenom;

            istr >> Prenom;

            m_Prenom = CPrenom (Prenom);

        } // if()

    } // if ()

    Normaliser ();
}

//////////
//          //
// Accesseur //
//          //
//////////

std::string CIdent::GetNom (void) const throw ()
{
    return m_Nom.CNom::GetNom();
} // GetNom ()

string CIdent::GetPrenom (void) const throw ()
{
    return m_Prenom.CPrenom::GetPrenom();
} // GetPrenom ()

```

```

string CIdent::GetAcronyme (void) throw ()
{
    return m_Acronyme;
} // GetAcronyme ()

string CIdent::NomPrenom (void) const throw ()
{
    return (m_Nom.CNom::GetNom() + ' ' +
           m_Prenom.CPrenom::GetPrenom());
} // NomPrenom ()

string CIdent::PrenomNom (void) const throw ()
{
    return m_Prenom.CPrenom::GetPrenom () + ' '
           + m_Nom.CNom::GetNom ();
} // PrenomNom ()

string CIdent::PNom (void) const throw ()
{
    return m_Prenom.GetDebutPrenom (1) + ". " + m_Nom.GetNom();
} // PNom ()

////////////////////////////////////
//                               //
// Opérateurs (classés selon    //
// la logique du code de Gray) //
//                               //
////////////////////////////////////

bool CIdent::operator < (const CIdent & Ident) throw ()
{
    return m_Acronyme < Ident.m_Acronyme;
} // operator <

ostream & operator << (ostream & flux, const CIdent & Ident)
{
    return flux << Ident.m_Acronyme;
} // operator <<

```

```

bool CIdent::operator <= (const CIdent & Ident) throw ()
{
    return m_Acronyme <= Ident.m_Acronyme;
} // operator <=

bool CIdent::operator == (const CIdent & Ident) throw ()
{
    return m_Acronyme == Ident.m_Acronyme;
} // operator ==

bool CIdent::operator != (const CIdent & Ident) throw ()
{
    return m_Acronyme != Ident.m_Acronyme;
} // operator !=

bool CIdent::operator >= (const CIdent & Ident) throw ()
{
    return m_Acronyme >= Ident.m_Acronyme;
} // operator >=

bool CIdent::operator > (const CIdent & Ident) throw ()
{
    return m_Acronyme > Ident.m_Acronyme;
} // operator >

bool EstStrIdent (const CString & Str) throw ()
{
    return Str.EstChaineAlphaAccentEspace()
        &&Str.NbreMots () == 2;
} // EstStrIdent ()

} // nsMonSpace

#endif

```

XI. CCpt.h

```

/**
 *
 * @ File : CCpt.h
 *
 * @ Authors : V.Castille M.Lapeyre
 *
 * @ Date : 11/03/2005
 *
 **/

#ifndef __CCPT_H__
#define __CCPT_H__

namespace nsMonSpace
{
    class CCpt
    {
        private:
            unsigned m_Cpt;

        public:

            ////////////////////////////////////////////////////
            //                               //
            // Constructeurs //
            //                               //
            ////////////////////////////////////////////////////

            CCpt (const unsigned i = 0) throw();

            ////////////////////////////////////////////////////
            //                               //
            // Accesseur //
            //                               //
            ////////////////////////////////////////////////////

            unsigned GetCpt (void) const throw();
    }
}

```

```
//////////  
//          //  
// Traitements //  
//          //  
//////////  
  
void Incr (void) throw();  
void Decr (void) throw();  
  
inline bool operator < (CCpt compteur2)  
{  
    return m_Cpt < compteur2.m_Cpt;  
}  
  
};//CCpt  
  
};//nsMonSpace  
  
#endif
```

XII. CCpt.cxx

```

/**
 *
 * @ File : CCpt.cxx
 *
 * @ Authors : V.Castille M.Lapeyre
 *
 * @ Date : 11/03/2005
 *
 **/

#if !defined __CCPT_CXX__
#define __CCPT_CXX__

#include "CCpt.h"

namespace nsMonSpace
{
    CCpt::CCpt (const unsigned i) throw ()
        :m_Cpt(i)
    {}

    unsigned CCpt::GetCpt (void) const throw ()
    {
        return m_Cpt;
    } // GetCpt ()

    void CCpt::Incr (void) throw ()
    {
        if (m_Cpt >= 0)
            ++ m_Cpt;
    } // Incr ()

    void CCpt::Decr (void) throw ()
    {
        if (m_Cpt >= 1)
            -- m_Cpt;
    } // Decr ()

} // nsMonSpace

#endif

```

XIII. CPerso.cxx

```

/**
 *
 * @ File : CPerso.cxx
 *
 * @ Authors : V.Castille, M.Lapeyre
 *
 * @ Date : 11/03/04
 *
**/

#if !defined __CPERSO_CXX__
#define __CPERSO_CXX__

#include "CPerso.h"
#include "CCpt.h"
#include "CString.h"

#include <sstream>
#include <cctype>

using namespace std;

namespace nsMonSpace
{
    ////////////////////////////////////
    //                               //
    // Constructeur //
    //                               //
    ////////////////////////////////////

    CPerso::CPerso (const CString & NomPrenom,
                    const unsigned Compteur,
                    const unsigned Classement) throw ()
        :CIdent (NomPrenom),
          m_Compteur (Compteur),
          m_Classement (Classement)
    {}
}

```

```

////////////////////////////////////
//                               //
// Accesseurs //
//                               //
////////////////////////////////////

CCpt CPerso::GetCompteur (void) const throw()
{
    return m_Compteur;

} // GetCompteur ()

unsigned CPerso::GetClassement (void) const throw()
{
    return m_Classement;

} // GetClassement ()

////////////////////////////////////
//                               //
// Traitement //
//                               //
////////////////////////////////////

void CPerso::IncrCompteur (void) throw()
{
    m_Compteur.Incr();

} // IncrCompteur ()

std::string CPerso::PersoToString (void) const throw()
{
    ostringstream Sortie;

    Sortie << NomPrenom() << ' '
           << m_Compteur.GetCpt() << ' ' << m_Classement;

    return Sortie.str();

} // PersoToString ()

```

```

CPerso StringToPerso (const CString & Str) throw ()
{
    istringstream is (Str);

    string Nom, Prenom;

    is >> Nom >> Prenom;

    unsigned Compteur, Classement;

    is >> Compteur >> Classement;

    string Temporaire;

    Temporaire += Nom;
    Temporaire += ' ';
    Temporaire += Prenom;

    return CPerso (Temporaire, Compteur, Classement);
} // StringToPerso ()

////////////////////
//                //
// Opérateurs //
//                //
////////////////////

bool CPerso::operator < (const CPerso & Perso) const throw ()
{
    return m_Classement < Perso.m_Classement;
} // operator <

ostream & operator << (ostream & flux, const CPerso & Perso)
{
    return flux << Perso.PersoToString();
} // operator <<

bool CPerso::operator <= (const CPerso & Perso) const throw ()
{
    return m_Classement <= Perso.m_Classement;
} // operator <=

```

```

bool CPerso::operator != (const CPerso & Perso) const throw ()
{
    return m_Classement != Perso.m_Classement;
} // operator !=

bool CPerso::operator == (const CPerso & Perso) const throw ()
{
    return m_Classement == Perso.m_Classement;
} // operator ==

bool CPerso::operator >= (const CPerso & Perso) const throw ()
{
    return m_Classement >= Perso.m_Classement;
} // operator >=

bool CPerso::operator > (const CPerso & Perso) const throw ()
{
    return m_Classement > Perso.m_Classement;
} // operator >

//////////
//          //
// Modifieur //
//          //
//////////

void CPerso::SetClassement (const unsigned Nouveau) throw()
{
    m_Classement = Nouveau;
} // SetClassement ()

} // namespace nsMonSpace ()

#endif

```

XIV. Acro1.cxx

```

/**
 * @ File : Acro1.cxx
 *
 * @ Authors : V.Castille, M.Lapeyre
 *
 * @ Date : 12/03/05
 *
**/

#include <iostream>
#include <fstream>
#include <exception>
#include <iomanip>
#include <string>
#include <vector>
#include <cctype>

#include "unistd.h"

#include "Char.h"
#include "CString.h"
#include "CPrenom.h"
#include "CIdent.h"
#include "CCpt.h"
#include "CPerso.h"
#include "nsUtil.h"

using namespace std;
using namespace nsMonSpace;
using namespace nsUtil;

namespace
{
    //////////////////////////////////////
    //                                     //
    // Initialisation des Variables //
    //                                     //
    //////////////////////////////////////

    typedef vector <CPerso*> CVecPtrPerso;
    typedef CVecPtrPerso::size_type CInd_t;

    const unsigned CstTailleMax (100);

    CVecPtrPerso VecPtrStock, VecPtrClass;

```

```

CIdent Ident;
CCpt Cpt;
unsigned Classement;

//////////
//          //
// Recherche //
//          //
//////////

CInd_t PosDicho (const CString & Acro)
{
    CInd_t Taille (VecPtrStock.size ());

    if (0 == Taille)
        return 0;

    if (Acro > *(VecPtrStock [Taille - 1])).GetAcronyme ())
        return Taille;

    CInd_t Deb (0), Fin (Taille -1);

    for ( ; Deb != Fin; )
    {
        CInd_t Milieu = (Deb + Fin) / 2;

        if (Acro > *(VecPtrStock [Milieu])).GetAcronyme())
            Deb = Milieu + 1;
        else
            Fin = Milieu;
    }//for

    return Deb;
} // PosDicho

```

```

////////////////////////////////////
//                               //
// Visualisation //
//                               //
////////////////////////////////////

void VisualisationStock (void)
{
    cout << "Ensemble des personnes (sans tri) :\n\n";

    for (CInd_t i (0); i < VecPtrStock.size (); ++i)
        cout << *(VecPtrStock [i]) << '\n';

    cout << endl;

    sleep (5);

} // VisualisationStock ()

void VisualisationClass (void)
{
    cout << "\nEnsemble des personnes"
        << " (dans l'ordre de classement) :\n\n";

    for (CInd_t i (0) ; i < VecPtrClass.size () ; ++i)
        cout << *VecPtrClass [i] << '\n';

    cout << endl;

    sleep (5);

} // VisualisationClass ()

```

```

//////////////////////////////////
//                               //
// Traitement //
//                               //
//////////////////////////////////

void Menu (void) throw ()
{
    cout << "\033[2J\033[1:1H";

    cout << "\n//////////////////////////////////\n"
         << "//                               //\n"
         << "// Devoir C++ : Acronyme 1 //\n"
         << "//                               //\n"
         << "//////////////////////////////////\n\n\n"
         << "Liste des opérations réalisable :\n\n"
         << "1. Augmentation de la moyenne d'une personne\n"
         << "2. Voir le classement d'une personne\n"
         << "3. Voir la personne qui a un classement donné\n"
         << "4. Quitter\n\n"
         << "Quel est votre choix : ";

} // Menu ()

void Traitement1 () throw ()
{
    cout << "\nVeuillez entrer un acronyme : ";
    CString Acro;
    Acro.LireString (EstCharAlphaNum);

    cout << endl;

    Acro.ToMajuscule ();

    CInd_t Pos (PosDicho (Acro));

    if (Pos == VecPtrStock.size() || Pos == 0)
    {
        cout << "\nAcronyme absent\n";
    } // if ()
}

```

```

else
{
    CPerso *PtrCourant = VecPtrStock [Pos];

    if ((*PtrCourant).GetAcronyme () != Acro)
    {
        cout << "\nAcronyme absent\n";
    } // if ()

    (*PtrCourant).IncrCompteur ();

    unsigned ClassCourant
        ((*PtrCourant).GetClassement ());

    if (ClassCourant != 1)
    {
        CPerso *PtrPreced
            (VecPtrClass [ClassCourant - 2]);

        unsigned ClassPreced
            ((*PtrPreced).GetClassement());

        if ((*PtrPreced).GetCompteur () <
            (*PtrCourant).GetCompteur ())
        {
            (*PtrCourant).SetClassement (ClassPreced);
            (*PtrPreced).SetClassement (ClassCourant);

            swap (VecPtrClass [ClassCourant - 1],
                VecPtrClass [ClassPreced]);
        } // if ()
    } // if ()

    cout << "\nLe score de " << Acro
        << " a été augmenté à "
        << *PtrCourant << "\n\n";

} //else

```

```

for ( ; ; )
{
    cout << "\nVoulez vous continuer (O/N) ? ";

    char Answer;

    cin >> Answer;

    Answer = toupper (Answer);

    if ('O' != Answer && 'N' != Answer)
    {
        cout << "\n\nChoix erroné,"
            << " veuillez recommencer...\n";
        continue;

    } // if ()

    if ('O' == Answer)
        Traitement1 ();
    else
    {
        VisualisationClass ();
        break;

    } //else

} // for ()

} // Traitement1 ()

void Traitement2 () throw ()
{
    cout << "\nVeuillez entrer un acronyme : ";
    CString Acro;
    Acro.LireString (EstCharAlphaNum);

    cout << endl;

    Acro.ToMajuscule ();

    CInd_t Pos (PosDicho (Acro));

    if (Pos == VecPtrStock.size() || Pos == 0)
    {
        cout << "\nAcronyme absent\n\n";

    } // if ()

```

```

else
{
    CPerso *PtrCourant = VecPtrStock [Pos];

    if ((*PtrCourant).GetAcronyme () != Acro)
    {
        cout << "\nAcronyme absent\n\n";

    } // if ()

    cout << '\n' << Acro << " est classé "
        << (*PtrCourant).GetClassement () << "\n\n";
} //else

for ( ; ; )
{
    cout << "Voulez vous continuer (O/N) ? ";

    char Answer;

    cin >> Answer;
    Answer = toupper (Answer);

    if ('O' != Answer && 'N' != Answer)
    {
        cout << "\nChoix erroné,"
            << " veuillez recommencer...\n\n";
        continue;

    } // if ()

    if ('N' == Answer)
    {
        VisualisationClass ();
        break;

    } // if ()

    if ('O' == Answer)
        Traitement2 ();

} // for ()

} // Traitement2 ()

```

```

void Traitement3 () throw ()
{
    cout << "\nVeuillez saisir un classement : ";

    CString StrClass;

    StrClass.LireString (EstCharNum);

    unsigned Class (StringToUnsigned(StrClass));

    if (0 == Class || Class > VecPtrStock.size())
        cout << "\n\nErreur de saisie\n\n";
    else
        if (VecPtrStock.size() == 0)
            cout << "\nPas d'acronyme pour"
                << " le classement demandé\n";
        else
        {
            cout << endl;
            cout << "\nLa personne dont le"
                << " classement est " << StrClass
                << " est " << *VecPtrClass [Class - 1]
                << "\n\n";

        } // else()

    for ( ; ; )
    {
        cout << "Voulez vous continuer (O/N) ? ";

        char Answer;

        cin >> Answer;

        Answer = toupper (Answer);

        if ('O' != Answer && 'N' != Answer)
        {
            cout << "\nChoix erroné,"
                << " veuillez recommencer...\n\n";
            continue;

        } // if ()
    }
}

```

```

        if ('N' == Answer)
        {
            VisualisationClass ();
            break;

        } // if ()

        if ('O' == Answer)
            Traitement3 ();

    } // for ()

} // Traitement3 ()

void FctAcro ()
{
    VecPtrStock.reserve (CstTailleMax);
    VecPtrClass.reserve (CstTailleMax);

    string Str;
    ifstream is ("PersoIn");

    for ( ; getline (is,Str) ; )
    {
        CPerso *Ptr = new CPerso();
        *Ptr = StringToPerso (Str);
        VecPtrStock.push_back (Ptr);

    } // for ()

    VisualisationStock ();
    unsigned Taille (VecPtrStock.size ());

    VecPtrClass.resize (Taille);

    for (unsigned i (0) ; i < Taille ; ++i)
        VecPtrClass [(VecPtrStock [i]).GetClassement () - 1]
            = VecPtrStock [i];

    CString Acro;

```

```

for ( ; ; )
{
    Menu ();

    char Choix;

    cin >> Choix;

    switch (Choix)
    {
        case '1' :
            Traitement1 ();
            Menu ();
            break;

        case '2' :
            Traitement2 ();
            Menu ();
            break;

        case '3' :
            Traitement3 ();
            Menu ();
            break;

        case '4' :
            cout << "\nMerci d'avoir"
                << " utilisé notre application."
                << " A très bientôt :)\n";
            sleep (3);
            cout << "\033[2J\033[1:1H";
            return;

        default :
            cout << "\nChoix invalide,"
                << " veuillez recommencer...\n";
            sleep(3);
            Menu ();

    } // switch ()

} // for ()

```

```

        // Sauvegarde

        ofstream os ("PersoOut");

        for (unsigned i (0) ; i < VecPtrStock.size () ; ++i)
            os << *VecPtrStock [i] << '\n';

        cout << endl;

    } // FctAcro ()

} // namespace

int main (void)
{
    enum { CstNoExc = 0, CstExcStd = 254, CstExcInconnue = 255 };

    try
    {
        cout << "\nACRONYME V.1 Devoir C++ 2005\n\n";
        sleep (2);
        cout << "\033[2J\033[1:1H";
        FctAcro ();
        return CstNoExc;

    } // try ()

    catch (const exception & Exc)
    {
        cerr << "Exception standard ou dérivée : "
              << Exc.what () << endl;

        return CstExcStd;

    } // catch ()

    catch (...)
    {
        cerr << "Exception inconnue" << endl;

        return CstExcInconnue;

    } // catch ()

} // main ()

```

XV. Acro2.cxx

```
/**
 * @ File : Acro2.cxx
 *
 * @ Authors : V.Castille, M.Lapeyre
 *
 * @ Date : 13/03/05
 *
 * @ MAJ réalisée à partir du fichier Acro1.cxx daté du 12/03/05
 *
 **/
```

```
#include <iostream>
#include <fstream>
#include <exception>
#include <iomanip>
#include <string>
#include <vector>
#include <cctype>
```

```
#include "unistd.h"
```

```
#include "Char.h"
#include "CString.h"
#include "CPrenom.h"
#include "CIdent.h"
#include "CCpt.h"
#include "CPerso.h"
#include "nsUtil.h"
```

```
using namespace std;
using namespace nsMonSpace;
using namespace nsUtil;
```

```
namespace
```

```
{
```

```
////////////////////////////////////
//                               //
// Initialisation des Variables //
//                               //
////////////////////////////////////
```

```
typedef vector <CPerso*> CVecPtrPerso;
typedef CVecPtrPerso::size_type CInd_t;
```

```
const unsigned CstTailleMax (100);
```

```
CVecPtrPerso VecPtrStock, VecPtrClass;
```

```

CIdent Ident;
CCpt Cpt;
unsigned Classement;

//////////
//          //
// Recherche //
//          //
//////////

CInd_t PosDicho (const CString & Acro)
{
    CInd_t Taille (VecPtrStock.size ());

    if (0 == Taille)
        return 0;

    if (Acro > *(VecPtrStock [Taille - 1])).GetAcronyme ())
        return Taille;

    CInd_t Deb (0), Fin (Taille -1);

    for ( ; Deb != Fin; )
    {
        CInd_t Milieu = (Deb + Fin) / 2;

        if (Acro > *(VecPtrStock [Milieu])).GetAcronyme())
            Deb = Milieu + 1;
        else
            Fin = Milieu;
    }//for

    return Deb;
} // PosDicho

```

```

////////////////////////////////////
//                               //
// Visualisation //
//                               //
////////////////////////////////////

void VisualisationStock (void)
{
    cout << "\nEnsemble des personnes (sans tri) :\n\n";

    for (CInd_t i (0); i < VecPtrStock.size (); ++i)
        cout << *(VecPtrStock [i]) << '\n';

    cout << endl;

    sleep (5);

} // VisualisationStock ()

void VisualisationClass (void)
{
    cout << "\nEnsemble des personnes"
        << " (dans l'ordre de classement) :\n\n";

    for (CInd_t i (0) ; i < VecPtrClass.size () ; ++i)
        cout << *VecPtrClass [i] << '\n';

    cout << endl;

    sleep (5);

} // VisualisationClass ()

```

```

////////////////////
//              //
// Traitement //
//              //
////////////////////

void Menu (void) throw ()
{
    cout << "\033[2J\033[1:1H";

    cout << "\n////////////////////////////////////////\n"
         << "//                               //\n"
         << "// Devoir C++ : Acronyme 2 //\n"
         << "//                               //\n"
         << "////////////////////////////////////////\n\n\n"
         << "Liste des opérations réalisable :\n\n"
         << "1. Augmentation de la moyenne d'une personne\n"
         << "2. Voir le classement d'une personne\n"
         << "3. Voir la personne qui a un classement donné\n"
         << "4. Ajout d'une nouvelle personne\n"
         << "5. Quitter\n\n"
         << "Quel est votre choix : ";

} // Menu ()

void Traitement1 () throw ()
{
    cout << "\nVeuillez entrer un acronyme : ";
    CString Acro;
    Acro.LireString (EstCharAlphaNum);

    cout << endl;

    Acro.ToMajuscule ();

    CInd_t Pos (PosDicho (Acro));

    if (Pos == VecPtrStock.size() || Pos == 0)
    {
        cout << "\nAcronyme absent\n";
    } // if ()
}

```

```

else
{
    CPerso *PtrCourant = VecPtrStock [Pos];

    if ((*PtrCourant).GetAcronyme () != Acro)
    {
        cout << "\nAcronyme absent\n";
    } // if ()

    (*PtrCourant).IncrCompteur ();

    unsigned ClassCourant
        ((*PtrCourant).GetClassement ());

    if (ClassCourant != 1)
    {
        CPerso *PtrPreced
            (VecPtrClass [ClassCourant - 2]);

        unsigned ClassPreced
            ((*PtrPreced).GetClassement());

        if ((*PtrPreced).GetCompteur () <
            (*PtrCourant).GetCompteur ())
        {
            (*PtrCourant).SetClassement (ClassPreced);
            (*PtrPreced).SetClassement (ClassCourant);

            swap (VecPtrClass [ClassCourant - 1],
                VecPtrClass [ClassPreced]);
        } // if ()
    } // if ()

    cout << "\nLe score de " << Acro
        << " a été augmenté à "
        << *PtrCourant << "\n\n";

} //else

```

```

for ( ; ; )
{
    cout << "\nVoulez vous continuer (O/N) ? ";

    char Answer;

    cin >> Answer;

    Answer = toupper (Answer);

    if ('O' != Answer && 'N' != Answer)
    {
        cout << "\n\nChoix erroné,"
            << " veuillez recommencer...\n";
        continue;

    } // if ()

    if ('O' == Answer)
        Traitement1 ();
    else
    {
        VisualisationClass ();
        break;

    } //else

} // for ()

} // Traitement1 ()

void Traitement2 () throw ()
{
    cout << "\nVeuillez entrer un acronyme : ";
    CString Acro;
    Acro.LireString (EstCharAlphaNum);

    cout << endl;

    Acro.ToMajuscule ();

    CInd_t Pos (PosDicho (Acro));

    if (Pos == VecPtrStock.size() || Pos == 0)
    {
        cout << "\nAcronyme absent\n\n";

    } // if ()

```

```

else
{
    CPerso *PtrCourant = VecPtrStock [Pos];

    if ((*PtrCourant).GetAcronyme () != Acro)
    {
        cout << "\nAcronyme absent\n\n";

    } // if ()

    cout << '\n' << Acro << " est classé "
        << (*PtrCourant).GetClassement () << "\n\n";

} //else

for ( ; ; )
{
    cout << "Voulez vous continuer (O/N) ? ";

    char Answer;

    cin >> Answer;
    Answer = toupper (Answer);

    if ('O' != Answer && 'N' != Answer)
    {
        cout << "\nChoix erroné,"
            << " veuillez recommencer...\n\n";
        continue;

    } // if ()

    if ('N' == Answer)
    {
        VisualisationClass ();
        break;

    } // if ()

    if ('O' == Answer)
        Traitement2 ();

} // for ()

} // Traitement2 ()

```

```

void Traitement3 () throw ()
{
    cout << "\nVeuillez saisir un classement : ";

    CString StrClass;

    StrClass.LireString (EstCharNum);

    unsigned Class (StringToUnsigned(StrClass));

    if (0 == Class || Class > VecPtrStock.size())
        cout << "\n\nErreur de saisie\n\n";
    else
        if (VecPtrStock.size() == 0)
            cout << "\nPas d'acronyme pour"
                << " le classement demandé\n";
        else
        {
            cout << endl;
            cout << "\nLa personne dont le"
                << " classement est " << StrClass
                << " est " << *VecPtrClass [Class - 1]
                << "\n\n";

        } // else()

    for ( ; ; )
    {
        cout << "Voulez vous continuer (O/N) ? ";

        char Answer;

        cin >> Answer;

        Answer = toupper (Answer);

        if ('O' != Answer && 'N' != Answer)
        {
            cout << "\nChoix erroné,"
                << " veuillez recommencer...\n\n";
            continue;

        } // if ()
    }
}

```

```

        if ('N' == Answer)
        {
            VisualisationClass ();
            break;
        } // if ()

        if ('O' == Answer)
            Traitement3 ();

    } // for ()

} // Traitement3 ()

void Traitement4 () throw()
{
    cout << "\nVeuillez saisir le Nom et"
         << " le Prenom de la nouvelle personne : ";

    CString StrNom;
    CString StrPrenom;
    CString Str;

    cin >> StrNom >> StrPrenom;

    Str = StrNom + ' ' + StrPrenom;

    if (!EstStrIdent (Str) || Str.size() == 0)
    {
        cout << "\nErreur de saisie\n\n";
    } // if ()
    else
    {
        CPerso *PtrNew = new CPerso (Str);

        CString NewAcronyme = PtrNew->GetAcronyme ();

        unsigned Taille = VecPtrStock.size();

        CInd_t Pos (PosDicho (NewAcronyme));

```

```

if (Pos != Taille && Pos != 0)
{
    delete PtrNew;

    cout << "\nL'acronyme est déjà"
         << " présent dans la base de donnée."
         << " L'ajout est impossible."
         << " Désolé.\n\n";

} // if ()
else
{
    VecPtrStock.insert
        (VecPtrStock.begin() + Pos, PtrNew);

    PtrNew->SetClassement (Taille + 1);

    cout << "\nLa personne ajoutée est : "
         << *PtrNew << endl;

} // else ()

} // else

for ( ; ; )
{
    cout << "\nVoulez vous continuer (O/N) ? ";

    char Answer;

    cin >> Answer;

    Answer = toupper (Answer);

    if ('O' != Answer && 'N' != Answer)
    {
        cout << "\nChoix erroné, veuillez"
             << " recommencer...\n\n";
        continue;

    } // if ()

```

```

        if ('N' == Answer)
        {
            VisualisationStock ();
            break;

        } // if ()

        if ('O' == Answer)
            Traitement4 ();

    } // for ()

} // Traitement4 ()

void Save (void)
{
    ofstream os ("PersoOut");

    for (unsigned i (0) ; i < VecPtrStock.size () ; ++i)
        os << *VecPtrStock [i] << '\n';

    cout << endl;

} //Save

void FctAcro ()
{
    VecPtrStock.reserve (CstTailleMax);
    VecPtrClass.reserve (CstTailleMax);

    string Str;
    ifstream is ("PersoIn");

    for ( ; getline (is,Str) ; )
    {
        CPerso *Ptr = new CPerso();
        *Ptr = StringToPerso (Str);
        VecPtrStock.push_back (Ptr);

    } // for ()

    VisualisationStock ();
    unsigned Taille (VecPtrStock.size ());

    VecPtrClass.resize (Taille);

```

```

for (unsigned i (0) ; i < Taille ; ++i)
    VecPtrClass [( *VecPtrStock [i]).GetClassement ()
                - 1] = VecPtrStock [i];

CString Acro;

for ( ; ; )
{
    Menu ();

    char Choix;

    cin >> Choix;

    switch (Choix)
    {
        case '1' :
            Traitement1 ();
            Menu ();
            break;

        case '2' :
            Traitement2 ();
            Menu ();
            break;

        case '3' :
            Traitement3 ();
            Menu ();
            break;

        case '4' :
            Traitement4 ();
            Menu ();
            break;

        case '5' :
            cout << "\nMerci d'avoir utilisé"
                << " notre application."
                << " A très bientôt :)\n";
            Save();
            sleep (3);
            cout << "\033[2J\033[1:1H";
            return;
    }
}

```

```

                default :
                    cout << "\nChoix invalide,"
                        << " veuillez recommencer...\n";
                    sleep(3);
                    Menu ();

                } // switch ()

            } // for ()

        } // FctAcro ()

    } // namespace

int main (void)
{
    enum { CstNoExc = 0, CstExcStd = 254, CstExcInconnue = 255 };

    try
    {
        cout << "\nACRONYME V.2 Devoir C++ 2005\n\n";
        sleep (2);
        cout << "\033[2J\033[1:1H";
        FctAcro ();
        return CstNoExc;

    } // try ()

    catch (const exception & Exc)
    {
        cerr << "Exception standard ou dérivée : "
            << Exc.what () << endl;

        return CstExcStd;

    } // catch ()

    catch (...)
    {
        cerr << "Exception inconnue" << endl;

        return CstExcInconnue;

    } // catch ()

} // main ()

```

XVI. Acro3.cxx

```
/**
 * @ File : Acro3.cxx
 *
 * @ Authors : V.Castille, M.Lapeyre
 *
 * @ Date : 14/03/05
 *
 * @ MAJ réalisée à partir du fichier Acro2.cxx daté du 13/03/05
 *
 **/
```

```
#include <iostream>
#include <fstream>
#include <exception>
#include <iomanip>
#include <string>
#include <vector>
#include <cctype>
```

```
#include "unistd.h"
```

```
#include "Char.h"
#include "CString.h"
#include "CPrenom.h"
#include "CIdent.h"
#include "CCpt.h"
#include "CPerso.h"
#include "nsUtil.h"
```

```
using namespace std;
using namespace nsMonSpace;
using namespace nsUtil;
```

```
namespace
```

```
{
```

```
////////////////////////////////////
//                               //
// Initialisation des Variables //
//                               //
////////////////////////////////////
```

```
typedef vector <CPerso*> CVecPtrPerso;
typedef CVecPtrPerso::size_type CInd_t;
```

```
const unsigned CstTailleMax (100);
```

```
CVecPtrPerso VecPtrStock, VecPtrClass;
```

```

CIdent Ident;
CCpt Cpt;
unsigned Classement;

//////////
//          //
// Recherche //
//          //
//////////

CInd_t PosDicho (const CString & Acro)
{
    CInd_t Taille (VecPtrStock.size ());

    if (0 == Taille)
        return 0;

    if (Acro > *(VecPtrStock [Taille - 1])).GetAcronyme ())
        return Taille;

    CInd_t Deb (0), Fin (Taille -1);

    for ( ; Deb != Fin; )
    {
        CInd_t Milieu = (Deb + Fin) / 2;

        if (Acro > *(VecPtrStock [Milieu])).GetAcronyme())
            Deb = Milieu + 1;
        else
            Fin = Milieu;
    }//for

    return Deb;
} // PosDicho

```

```

////////////////////////////////////
//                               //
// Visualisation //
//                               //
////////////////////////////////////

void VisualisationStock (void)
{
    cout << "\nEnsemble des personnes (sans tri) :\n\n";

    for (CInd_t i (0); i < VecPtrStock.size (); ++i)
        cout << *(VecPtrStock [i]) << '\n';

    cout << endl;

    sleep (5);

} // VisualisationStock ()

void VisualisationClass (void)
{
    cout << "\nEnsemble des personnes"
        << " (dans l'ordre de classement) :\n\n";

    for (CInd_t i (0) ; i < VecPtrClass.size () ; ++i)
        cout << *VecPtrClass [i] << '\n';

    cout << endl;

    sleep (5);

} // VisualisationClass ()

```

```

//////////////////////////////////
//                               //
// Traitement //
//                               //
//////////////////////////////////

void Menu (void) throw ()
{
    cout << "\033[2J\033[1:1H";

    cout << "\n////////////////////////////////////////\n"
        << "//                               //\n"
        << "// Devoir C++ : Acronyme 3 //\n"
        << "//                               //\n"
        << "////////////////////////////////////////\n\n\n"
        << "Liste des opérations réalisable :\n\n"
        << "1. Augmentation de la moyenne d'une personne\n"
        << "2. Voir le classement d'une personne\n"
        << "3. Voir la personne qui a un classement donné\n"
        << "4. Ajout d'une nouvelle personne\n"
        << "5. Suppression d'une personne\n"
        << "6. Quitter\n\n"
        << "Quel est votre choix : ";

} // Menu ()

void Traitement1 () throw ()
{
    cout << "\nVeuillez entrer un acronyme : ";
    CString Acro;
    Acro.LireString (EstCharAlphaNum);

    cout << endl;

    Acro.ToMajuscule ();

    CInd_t Pos (PosDicho (Acro));

    if (Pos == VecPtrStock.size() || Pos == 0)
    {
        cout << "\nAcronyme absent\n";
    } // if ()
}

```

```

else
{
    CPerso *PtrCourant = VecPtrStock [Pos];

    if ((*PtrCourant).GetAcronyme () != Acro)
    {
        cout << "\nAcronyme absent\n";
    } // if ()

    (*PtrCourant).IncrCompteur ();

    unsigned ClassCourant
        ((*PtrCourant).GetClassement ());

    if (ClassCourant != 1)
    {
        CPerso *PtrPreced
            (VecPtrClass [ClassCourant - 2]);

        unsigned ClassPreced
            ((*PtrPreced).GetClassement());

        if ((*PtrPreced).GetCompteur () <
            (*PtrCourant).GetCompteur ())
        {
            (*PtrCourant).SetClassement (ClassPreced);
            (*PtrPreced).SetClassement (ClassCourant);

            swap (VecPtrClass [ClassCourant - 1],
                VecPtrClass [ClassPreced]);
        } // if ()
    } // if ()

    cout << "\nLe score de " << Acro
        << " a été augmenté à "
        << *PtrCourant << "\n\n";

} //else

```

```

for ( ; ; )
{
    cout << "\nVoulez vous continuer (O/N) ? ";

    char Answer;

    cin >> Answer;

    Answer = toupper (Answer);

    if ('O' != Answer && 'N' != Answer)
    {
        cout << "\n\nChoix erroné,"
            << " veuillez recommencer...\n";
        continue;

    } // if ()

    if ('O' == Answer)
        Traitement1 ();
    else
    {
        VisualisationClass ();
        break;

    } //else

} // for ()

} // Traitement1 ()

void Traitement2 () throw ()
{
    cout << "\nVeuillez entrer un acronyme : ";
    CString Acro;
    Acro.LireString (EstCharAlphaNum);

    cout << endl;

    Acro.ToMajuscule ();

    CInd_t Pos (PosDicho (Acro));

    if (Pos == VecPtrStock.size() || Pos == 0)
    {
        cout << "\nAcronyme absent\n\n";

    } // if ()

```

```

else
{
    CPerso *PtrCourant = VecPtrStock [Pos];

    if ((*PtrCourant).GetAcronyme () != Acro)
    {
        cout << "\nAcronyme absent\n\n";

    } // if ()

    cout << '\n' << Acro << " est classé "
        << (*PtrCourant).GetClassement ()
        << "\n\n";

} //else

for ( ; ; )
{
    cout << "Voulez vous continuer (O/N) ? ";

    char Answer;

    cin >> Answer;
    Answer = toupper (Answer);

    if ('O' != Answer && 'N' != Answer)
    {
        cout << "\nChoix erroné,"
            << " veuillez recommencer...\n\n";
        continue;

    } // if ()

    if ('N' == Answer)
    {
        VisualisationClass ();
        break;

    } // if ()

    if ('O' == Answer)
        Traitement2 ();

} // for ()

} // Traitement2 ()

```

```

void Traitement3 () throw ()
{
    cout << "\nVeuillez saisir un classement : ";

    CString StrClass;

    StrClass.LireString (EstCharNum);

    unsigned Class (StringToUnsigned(StrClass));

    if (0 == Class || Class > VecPtrStock.size())
        cout << "\n\nErreur de saisie\n\n";
    else
        if (VecPtrStock.size() == 0)
            cout << "\nPas d'acronyme pour"
                << " le classement demandé\n";
        else
        {
            cout << endl;
            cout << "\nLa personne dont le"
                << " classement est " << StrClass
                << " est " << *VecPtrClass [Class - 1]
                << "\n\n";

        } // else()

    for ( ; ; )
    {
        cout << "Voulez vous continuer (O/N) ? ";

        char Answer;

        cin >> Answer;

        Answer = toupper (Answer);

        if ('O' != Answer && 'N' != Answer)
        {
            cout << "\nChoix erroné,"
                << " veuillez recommencer...\n\n";
            continue;

        } // if ()
    }
}

```

```

        if ('N' == Answer)
        {
            VisualisationClass ();
            break;
        } // if ()

        if ('O' == Answer)
            Traitement3 ();

    } // for ()

} // Traitement3 ()

void Traitement4 () throw()
{
    cout << "\nVeuillez saisir le Nom"
         << " et le Prenom de la nouvelle personne : ";

    CString StrNom;
    CString StrPrenom;
    CString Str;

    cin >> StrNom >> StrPrenom;

    Str = StrNom + ' ' + StrPrenom;

    if (!EstStrIdent (Str) || Str.size() == 0)
    {
        cout << "\nErreur de saisie\n\n";
    } // if ()
    else
    {
        CPerso *PtrNew = new CPerso (Str);

        CString NewAcronyme = PtrNew->GetAcronyme ();

        unsigned Taille = VecPtrStock.size();

        CInd_t Pos (PosDicho (NewAcronyme));

```

```

if (Pos != Taille && Pos != 0)
{
    delete PtrNew;

    cout << "\nL'acronyme est déjà"
         << " présent dans la base de donnée."
         << " L'ajout est impossible."
         << " Désolé.\n\n";

} // if ()
else
{
    VecPtrStock.insert
        (VecPtrStock.begin() + Pos, PtrNew);

    PtrNew->SetClassement (Taille + 1);

    cout << "\nLa personne ajoutée est : "
         << *PtrNew << endl;

} // else ()

} // else

for ( ; ; )
{
    cout << "\nVoulez vous continuer (O/N) ? ";

    char Answer;

    cin >> Answer;

    Answer = toupper (Answer);

    if ('O' != Answer && 'N' != Answer)
    {
        cout << "\nChoix erroné,"
             << " veuillez recommencer...\n\n";
        continue;

    } // if ()
}

```

```

        if ('N' == Answer)
        {
            VisualisationStock ();
            break;

        } // if ()

        if ('O' == Answer)
            Traitement4 ();

    } // for ()

} // Traitement4 ()

void Traitement5 ()
{
    cout << "\nVeuillez saisir l'acronyme"
          << " de la personne à supprimer : ";

    CString Acro;

    Acro.LireString (EstCharAlphaNum);

    cout << endl;

    Acro.ToMajuscule ();

    CInd_t Pos (PosDicho (Acro));

    unsigned Taille = VecPtrStock.size();

    if (Pos == Taille)
    {
        cout << "\nAcronyme absent\n";

    } // if()

```

```

else
{
    CPerso *PtrCourant (VecPtrStock [Pos]);

    if ((*PtrCourant).GetAcronyme() != Acro)
    {
        cout << "\nAcronyme absent\n";
    } // if ()

    if ((*PtrCourant).GetClassement () == 0)
    {
        cout << "\nLa personne est déjà supprimée\n";
    } // if ()
    else
    {
        cout << "\nLa personne à supprimer"
            << " de la liste est "
            << *PtrCourant << endl;

        cout << "\nConfirmez-vous la"
            << " suppression (0/N) : ";
        CString Choix;
        cin >> Choix;
        Choix.ToMajuscule ();

        if ("0" != Choix)
        {
            cout << "\nSuppression annulée\n";
        } // if ()
        else
        {
            CInd_t IndClass
                = PtrCourant->GetClassement () -1;

            PtrCourant->SetClassement (0);
        } // else ()
    } // else ()
} // else ()

```

```

for ( ; ; )
{
    cout << "\nVoulez vous continuer (O/N) ? ";

    char Answer;

    cin >> Answer;

    Answer = toupper (Answer);

    if ('O' != Answer && 'N' != Answer)
    {
        cout << "\nChoix erroné,"
            << " veuillez recommencer...\n\n";
        continue;

    } // if ()

    if ('N' == Answer)
    {
        VisualisationStock ();
        break;

    } // if ()

    if ('O' == Answer)
        Traitement5 ();

    } // for ()

} // Traitement5 ()

void Save (void)
{
    ofstream os ("PersoOut");

    for (unsigned i (0) ; i < VecPtrStock.size () ; ++i)
        os << *VecPtrStock [i] << '\n';

    cout << endl;

} //Save

```

```

void Maj (void)
{
    unsigned Nb = 1;

    for (CInd_t i = 0; i < VecPtrClass.size(); ++i)
        if (VecPtrClass [i]->GetClassement () != 0)
            VecPtrClass [i]->SetClassement (Nb++);

    VisualisationStock ();
    cout << endl;

    VisualisationClass ();
    cout << endl;

} // Maj ()

void FctAcro ()
{
    VecPtrStock.reserve (CstTailleMax);
    VecPtrClass.reserve (CstTailleMax);

    string Str;
    ifstream is ("PersoIn");

    for ( ; getline (is,Str) ; )
    {
        CPerso *Ptr = new CPerso();
        *Ptr = StringToPerso (Str);
        VecPtrStock.push_back (Ptr);

    } // for ()

    VisualisationStock ();
    unsigned Taille (VecPtrStock.size ());

    VecPtrClass.resize (Taille);

    for (unsigned i (0) ; i < Taille ; ++i)
        VecPtrClass [(VecPtrStock [i]).GetClassement ()
                    - 1] = VecPtrStock [i];

    CString Acro;

```

```

for ( ; ; )
{
    Menu ();

    char Choix;

    cin >> Choix;

    switch (Choix)
    {
        case '1' :
            Traitement1 ();
            Menu ();
            break;

        case '2' :
            Traitement2 ();
            Menu ();
            break;

        case '3' :
            Traitement3 ();
            Menu ();
            break;

        case '4' :
            Traitement4 ();
            Menu ();
            break;

        case '5' :
            Traitement5 ();
            Maj();
            Menu ();
            break;

        case '6' :
            cout << "\nMerci d'avoir"
                << " utilisé notre application."
                << " A très bientôt :)\n";
            Save();
            sleep (3);
            cout << "\033[2J\033[1:1H";
            return;
    }
}

```

```

                default :
                    cout << "\nChoix invalide,"
                        << " veuillez recommencer...\n";
                    sleep(3);
                    Menu ();

                } // switch ()

            } // for ()

        } // FctAcro ()

    } // namespace

int main (void)
{
    enum { CstNoExc = 0, CstExcStd = 254, CstExcInconnue = 255 };

    try
    {
        cout << "\nACRONYME V.3 Devoir C++ 2005\n\n";
        sleep (2);
        cout << "\033[2J\033[1:1H";
        FctAcro ();
        return CstNoExc;

    } // try ()

    catch (const exception & Exc)
    {
        cerr << "Exception standard ou dérivée : "
            << Exc.what () << endl;

        return CstExcStd;

    } // catch ()

    catch (...)
    {
        cerr << "Exception inconnue" << endl;

        return CstExcInconnue;

    } // catch ()

} // main ()

```