Responsable pédagogique	AF	**AM**	OP	PM
Période	Sem1	**Sem2**	Sem3	Sem4
Volume horaire	Cours/TD		TP	
			6	

[CPP1]
DU C VERS LA POO...

Indicateur temporel (hors rédaction du compte-rendu) :

questions	1h	2h	3h	4h	5h	6h
1 .. 6						
7 .. 14						
15 .. 18						
19 .. 22						
23 .. 25						
26 .. 28						

Objectifs : Passage en douceur du C vers les premiers concepts de Programmation Orientée Objet.

Documents à rendre : Compte-rendu contenant à minima les sources (avec entête standard) et les résultats obtenus.

Cahier des Charges : En vue de la gestion d'un fichier de membres d'un club sportif, on souhaite dans un premier temps manipuler des objets représentatifs des adhérents. Un tel objet, pour les besoins de l'exercice, ne contiendra ici que le nom, le prénom et la date de naissance de la personne.

Création d'un type « date » personnalisé

Le format doit tenir compte de la localisation (France) : par exemple « 01/04/2011 » ou « 1er avril 2011 ». Il faut donc définir les types de données permettant de regrouper les informations de jour, mois, année et de manipuler les mois en lettres.
Commençons par les 12 mois du calendrier...

1. Dans un nouveau répertoire de travail, créez deux nouveaux fichiers nommés `tmonth.h` et `tmonth.cpp` (remarquez l'extension, nous allons bientôt utiliser le compilateur C++...).

2. Remplir le fichier *header* avec la définition d'un type énuméré `TMonth` :

```
#ifndef TMONTH_H
#define TMONTH_H

typedef enum { JAN=1, FEV, MAR, AVR, MAI, JUI,
               JUL, AOU, SEP, OCT, NOV, DEC }  TMonth ;

#endif
```

3. Créez une fonction dédiée nommée `strMonth()` capable de recevoir en argument une valeur de type `TMonth` et de retourner un pointeur sur une chaîne constante contenant le mois en toutes lettres.
   Prototype : `const char* strMonth(TMonth month ) ;`
4. Complétez la fonction qui peut être implémentée dans `tmonth.cpp` de la manière suivante. Elle travaille avec un tableau de chaînes constantes :

```
#include "tmonth.h"

char  mm[12][10] = {
      "janvier",  "février",
      // TODO : compléter la liste…
      "décembre"
} ;

const char* strMonth(TMonth month )
{
      // TODO : l'argument month doit être compris entre 1 et 12
      // TODO : retourner la chaîne adéquate…
      return mm[ ??? ] ;
}
```

5. Ajoutez un programme de test utilisateur nommé `main1.cpp` et contenant les lignes suivantes :

```
#include "tmonth.h"
#include <iostream>                                                    [1]
using namespace std ;                                                  [2]

int main(int argc, char** argv )
{
      for ( TMonth m = JAN ; m <= DEC ; m = (TMonth)(m + 1) ) {        [3]
            cout << strMonth(m) << endl ;                              [4]
      }
      return 0 ;
}
```

6. Compilez et testez le programme. La ligne de commande est la suivante :

```
g++ -Wall -o test1 main1.cpp tmonth.cpp
```

Le programme `test1` doit simplement afficher la liste des mois du calendrier.

Quelques explications s'imposent !

[1] Inclusion d'un fichier d'en-tête de la librairie standard du C++ qui remplace le traditionnel `stdio.h` du langage C (*iostream* = flux d'entrées/sorties).

L'absence d'extension `.h` caractérise la version moderne de la librairie, où toutes les ressources sont définies dans un espace de nommage nommé `std`.

[2] Nous spécifions au compilateur que nous voulons utiliser l'espace de nommage `std`. Sans cela, chaque référence à une ressource standard doit être précédée d'une information supplémentaire (voir plus loin…).

[3] Simple boucle destinée à balayer tous les mnémoniques du type énuméré `TMonth`. À noter la déclaration de la variable `m` seulement là où on en a besoin ; et le transtypage `TMonth` dans l'expression d'évolution : le compilateur C++ est beaucoup plus strict que son homologue C, il ne fait aucun *cast* implicite, il faut tout lui dire !

[4] Injection sur la sortie standard. Le flux `cout` remplace le canal `stdout` du C, `endl` est une version portable de `'\n'` ; l'opérateur `<<` est ici surchargé comme opérateur d'injection afin de permettre une écriture illustrant les données envoyées dans le « tuyau ».

Les identifiants `cout` et `endl` sont apportés par *iostream* ; mais sans la ligne *[2]*, cette utilisation du flux aurait dû s'écrire :

```
std::cout << strMonth(m) << std::endl ;
```

7. Nous pouvons maintenant créer notre type « date » proprement dit… Créez un fichier `tdate.h` et le remplir avec la définition du type `TDate` :

```
#ifndef TDATE_H
#define TDATE_H

#include "tmonth.h"

typedef struct {
      unsigned    day ;
      TMonth      month ;
      unsigned    year ;
} TDate ;

#endif
```

8. Créez un programme de test `main2.cpp` contenant le code suivant :

```
#include "tdate.h"
#include <iostream>
using namespace std ;

int main(int argc, char** argv )
{
      TDate tdate1 = { 14, JUL, 1789 } ;                              [5]
      TDate tdate2 ;                                                  [6]

      unsigned j, m, a ;
      cout << "jour, mois et année (séparés par espaces) ? " ;
      cin >> j >> m >> a ;                                            [7]

      tdate2.day = j ;                                                [8]
      tdate2.month = (TMonth)m ;
      tdate2.year = a ;

      cout << "date 1 : " << tdate1.day << '/' ;                      [9]
      cout << tdate1.month << '/' << tdate1.year << endl ;
      cout << "date 2 : " << tdate2.day << '/' ;
      cout << strMonth(tdate2.month) << '/' << tdate2.year << endl ;

      return 0 ;
}
```

[5] Déclaration et initialisation d'un objet de type structuré `TDate`.

[6] Déclaration d'un deuxième objet sans initialisation.

[7] Saisie de trois valeurs entières. Le flux `cin` représente l'entrée standard (canal `stdin` du C) ; l'opérateur >> est utilisé ici comme opérateur d'extraction.

[8] Affectation des champs de `tdate2` avec les valeurs saisies.

[9] Affichage des deux dates.

9. Générez le programme `test2` et testez-le…

   C'est là que les ennuis commencent ! Que ce passe-t-il si l'utilisateur rentre des valeurs aberrantes (valeur trop grande, valeur négative, …) ?

   « … il faut toujours se méfier de la chose bizarre située entre la chaise et le clavier… »

Pour durcir notre code, il faut interdire à l'utilisateur du type `TDate` de manipuler directement ses

champs, et le forcer à passer par des fonctions dédiées d'accès en lecture (*get*) et en écriture (*set*). Ces fonctions nécessitent forcément en argument une référence sur l'objet `TDate` concerné.

Exemple : pour le champ `month`, on peut imaginer les deux fonctions suivantes. L'accès en écriture permet de mettre en place les tests de validité de la valeur transmise :

```
TMonth getMonth(TDate* date )
{
      return date->month ;
}

void setMonth(TDate* date, TMonth month )
{
      if ( month < JAN )      month = JAN ;
      if ( month > DEC )      month = DEC ;
      date->month = month ;
}
```

10. En vous inspirant du modèle ci-dessus, ajoutez à `tdate.h` les 6 prototypes *get/set* nécessaires pour accès aux trois champs `day`, `month` et `year`.

11. Créez le fichier `tdate.cpp` afin d'implémenter ces fonctions. Le jour doit être limité dans un intervalle valide et l'année peut aussi subir un test de validité.

12. Modifiez le programme de test afin de remplacer les instructions du point *[8]*. Pour le mois, cela doit donc devenir : `setMonth(&tdate2, (TMonth)m ) ;`

13. Modifiez la ligne de commande *g++* et testez le programme. Ajoutez ensuite une fonction d'initialisation globale (qui permet d'initialiser les trois champs) ; pensez à réutiliser le code disponible :

```
void  setDate(TDate* date, unsigned day, TMonth month, unsigned year ) ;
```

14. Remplacez dans le programme de test l'initialisation de `tdate1` par un appel à la fonction précédente. Testez le programme.

Ces petites manipulations rendent compte d'une tentative de développement d'un programme robuste. Les données ne sont manipulées qu'au travers de fonctions qui leurs sont dédiées : ce concept porte le nom d'abstraction de données (l'utilisateur n'a plus besoin de savoir comment est défini en interne le type `TDate` à partir du moment où il dispose de la documentation des fonctions d'accès). Le fournisseur de `TDate` peut ainsi mettre en place toutes les sécurités utiles.

Malheureusement ici, rien n'empêche l'utilisateur de continuer à attaquer directement les champs de ses objets `TDate`...

Première classe

Pour un objet `TDate`, les données représentent son état et les fonctions définissent son comportement. La Programmation Orientée Objet permet de regrouper en une seule entité état et comportement : cette encapsulation prend la forme d'une classe.
Voyons ce que peut devenir notre type « date » en le définissant au moyen d'une classe...

15. Créez un nouveau fichier `cdate.h` contenant les lignes suivantes. Il s'agit de la définition d'une classe nommée `CDate` regroupant des données privées (que nous appellerons des attributs) et des fonctions publiques (que nous appellerons des méthodes) :

```
#ifndef CDATE_H
#define CDATE_H

#include "tmonth.h"

class CDate {

  private :
      unsigned    day ;
      unsigned    month ;
      unsigned    year ;

  public :                                                          [13]
      CDate() { set(1, JAN, 1900 ) ; }                              [14]

      CDate(unsigned day, TMonth month, unsigned year )             [15]
      {
            set(day, month, year ) ;
      }

      TMonth getMonth() const ;                                     [16]
      void   setMonth(TMonth month ) ;

      void   set(unsigned day, TMonth month, unsigned year ) ;      [17]
} ;

#endif
```

formalisme UML

CDate
-day : unsigned -month : unsigned -year : unsigned
+CDate() +CDate() +getMonth() : TMonth +setMonth() : void +set() : void

16. Ajoutez le fichier `cdate.cpp` pour implémenter les méthodes :

```
#include "cdate.h"

TMonth CDate::getMonth() const                                      [18]
{
      return (TMonth)month ;
}

void CDate::setMonth(TMonth month )
{
      if ( month < JAN )      month = JAN ;
      if ( month > DEC )      month = DEC ;
      this->month = (unsigned)month ;                               [19]
}

void CDate::set(unsigned day, TMonth month, unsigned year )
{
      // TODO
}
```

[10] `class` est un mot réservé du langage C++.

[11] Annonce de zone de déclarations privées (accessibles seulement par les membres de la classe) ; les attributs `day`, `month` et `year` ne seront pas visibles par l'utilisateur de la classe.

[12] Notez que le mois est ici codé en interne comme un entier non signé, alors qu'il sera vu comme un `TMonth` par les utilisateurs (encore un exemple d'abstraction de données).

[13] Annonce de zone de déclarations publiques (accessibles sans restriction, comme les champs d'une structure…).

[14] Méthode particulière sans type de retour et portant le même nom que la classe. Il s'agit du constructeur par défaut (sans argument) qui sera appelé automatiquement lors de toute création de variable (que nous appellerons instance dans le cas d'un « type » classe). Ce constructeur initialise l'objet avec une date arbitraire.

[15] Autre constructeur (le cumul n'est pas interdit !). Celui-ci va permettre de recevoir des valeurs d'initialisation lors de la déclaration d'instances.
Les constructeurs sont ici implémentés directement dans la déclaration de leur classe, on dit qu'ils sont *inline*.

[16] Exemples d'accesseurs. La méthode *get* est spécifiée `const` pour bien indiquer qu'elle ne modifie en rien l'objet.

[17] Méthode d'initialisation globale, utilisée notamment par les constructeurs.

[18] Implémentation externe (par opposition à *inline*) d'une méthode. Le nom complet de la méthode est constitué de l'identifiant de sa classe et de l'opérateur de portée **::** avant son nom propre.

[19] Le pointeur spécial `this` permet d'accéder au membre `month` de l'objet, ici pour qu'il n'y ait pas d'ambiguïté avec l'argument homonyme de la méthode.

17. Complétez la définition de la classe et son implémentation avec des accesseurs relatifs aux attributs `day` et `year`. Finalisez le corps de la méthode `set()` en pensant encore une fois à réutiliser le code disponible.

18. Créez un programme `test3` par le biais d'un nouveau fichier `main3.cpp`. La classe `CDate` peut être utilisée comme suit :

```
#include "cdate.h"

int main(int argc, char** argv )
{
      CDate cdate1 ;

      cdate1.set(14, JUL, 1789 ) ;

      CDate cdate2(11, SEP, 2001 ) ;

      // ... ...

      cdate2.setMonth( OCT ) ;
      cdate2.setYear( 2011 ) ;

      // ... ...

      return 0 ;
}
```

déclaration d'une instance cdate1 *: le constructeur par défaut est automatiquement invoqué*

appel de la méthode set() *appliquée à l'objet* cdate1 *(le séparateur est le point comme pour accéder aux champs de structure).*

déclaration d'une instance cdate2 *avec passage de valeurs au deuxième constructeur*

autres appels de méthodes...

Ligne de commande :

```
g++ -Wall -o test3 main3.cpp tmonth.cpp cdate.cpp
```

Dans le programme basé sur le type `TDate`, c'est l'utilisateur qui prenait en charge le format d'affichage des dates. Dans la version `CDate`, nous allons déléguer ces opérations à la classe...

19. Nous allons créer une nouvelle méthode publique nommée `print()`.
Le type de retour de cette méthode doit être une chaîne de caractères de classe `string` dans l'espace de nommage `std`. Les 2 lignes à ajouter en bonne place dans `cdate.h` sont donc :

```
#include <string>
std::string print() const ;
```

On souhaite que l'utilisateur puisse obtenir des affichages au format jj/mm/aaaa en écrivant des instructions du style :

```
cout << cdate1.print() << endl ;
cout << cdate2.print() << endl ;
```

20. Implémentez pour cela la méthode `print()` dans `cdate.cpp` de la manière suivante :

```
std::string CDate::print() const
{
	std::ostringstream	oss ;

	oss << std::setfill('0') ;
	oss << std::setw(2) << day << '/' ;
	oss << std::setw(2) << month << '/' ;
	oss << std::setw(4) << year ;
	return oss.str() ;
}
```

Ce code nécessite les inclusions `<sstream>` et `<iomanip>`. La classe `ostringstream` représente un flux de sortie de type *string*. La méthode `setfill()` spécifie le caractère de remplissage et la méthode `setw()` indique la largeur d'affichage.

En dernière ligne, la méthode `str()` récupère le *string* fabriqué sur le flux `oss` ; c'est la valeur de retour.

21. Testez le programme en affichant les valeurs des instances `cdate1` et `cdate2`.

22. Ajoutez maintenant une nouvelle ressource `printLong()` permettant d'obtenir une chaîne avec le mois en toutes lettres ; pensez à utiliser pour cela la fonction `strMonth()`. Vérifiez que le fonctionnement respecte exactement les cas suivants :

day	month	year	avec `print()`	avec `printLong()`
14	JUL	1789	14/07/1789	14 juillet 1789
1	AVR	2011	01/04/2011	1er avril 2011

Pour le programmeur qui utilise la classe `CDate`, le résultat est déjà intéressant… Mais pourquoi ne pas lui autoriser directement les écritures du style `cout << cdate1 << endl ;` ?

C'est possible en C++ ! Il suffit d'indiquer au compilateur comment il doit réagir lorsqu'il trouve un opérateur `<<` encadré par un flux `ostream` à gauche et un objet `CDate` à droite…

23. Dans `cdate.h`, ajouter l'inclusion `<ostream>` ; puis, <u>après la définition</u> de la classe `CDate`, ajoutez le prototype suivant :

```
std::ostream& operator << (std::ostream& s, const CDate& date ) ;
```

Cette fonction n'est pas membre de `CDate`. C'est une surcharge de l'opérateur normalement utilisé comme décalage à gauche (le mot `operator` est un mot réservé du langage C++).

Elle reçoit en argument les deux opérandes souhaités dans notre cas et retourne un flux de sortie identique à son argument de gauche (les symboles `&` après les types seront expliqués plus tard…).

24. Implémentez la fonction comme suit :

```
std::ostream& operator << (std::ostream& s, const CDate& date )
{
	s << date.print() ;
	return s ;
}
```

25. Testez le programme. L'usage de la surcharge précédente doit donner le même résultat que l'emploi de la méthode `print()` lorsqu'ils sont appliqués à des objets de classe `CDate`.
Pour l'utilisateur de la classe, l'écriture devient de plus en plus simple…

À vous de jouer !

Fort des expériences précédentes, vous allez maintenant vous occuper des membres de notre club sportif. Chaque individu sera modélisé par un objet de classe `CMembre` ; la liste des attributs traités sera volontairement minimaliste : le nom et le prénom (de classe `std::string`) et la date de naissance de classe `CDate`.

26. Préparez les nouveaux fichiers `cmembre.h`, `cmembre.cpp` et `main4.cpp` pour les tests.

27. Développez la classe `CMembre` avec au minimum :
    - les attributs privés `nom`, `prenom`, et `dateNaiss` ;
    - les accesseurs associés (en lecture et en écriture), implémentés en *inline* ;
    - un constructeur par défaut initialisant l'objet avec "?" pour les noms et prénoms et le 1er janvier 1900 pour la date de naissance ;
    - un deuxième constructeur acceptant nom et prénom en arguments ;
    - une méthode `print()` retournant les 3 attributs sous forme d'une unique chaîne.

    Les 3 dernières méthodes sont à implémenter en externe (donc dans `cmembre.cpp`).

28. Proposez un programme utilisateur de test permettant de valider le fonctionnement efficient de la classe `CMembre`.