Tous documents manuscrits de TD et de cours autorisés.
Support de cours autorisé.
Tous matériels électroniques éteints.
Durée 3 heures.

Exercice 1 : Les nombre aléatoires

Dans de nombreux cas, il est nécessaire de disposer de valeurs aléatoires. Pour cela, on utilise habituellement la bibliothèque <stdlib.h> et les deux fonctions suivantes :

- **int rand (void)**; cette fonction retourne un nombre aléatoire entier dans l'intervalle [0, RAND_MAX]. La constante RAND_MAX est fixée dans la bibliothèque <stdlib.h>. Il convient toutefois d'utiliser la fonction **srand** pour réinitialiser le générateur aléatoire et ainsi obtenir de « vrais » nombres aléatoires ;
- **void srand (unsigned int** *seed***)**; cette fonction permet de recaler le générateur aléatoire. Pour le réinitialiser (position de départ), il faut utiliser 1 comme argument. N'importe quelle autre valeur positionne le générateur à une position aléatoire. Classiquement, on utilise **srand((unsigned)time(NULL));** avec la bibliothèque <time.h> ce qui garantit que le prochain nombre généré est imprévisible (aléatoire !).

Je vous demande de développer deux classes pour manipuler en C++ les nombres aléatoires.

a. Ecrire une classe *NbAleat* qui permet de représenter un nombre entier positif aléatoire. Cette classe doit au moins posséder les méthodes suivantes :

- **NbAleat () ;** construction d'un nombre aléatoire dans l'intervalle [0, RAND_MAX]
- **NbAleat (const NbAleat &) ;** construction d'un nombre aléatoire par recopie d'un nombre aléatoire
- **virtual ~NbAleat () ;** destruction du nombre aléatoire
- **unsigned int Valeur () const ;** retourne la valeur aléatoire générée
- **operator unsigned int () const ;** convertit le nombre aléatoire en un entier non signé
- **void Regen () ;** retirage du nombre aléatoire dans l'intervalle [0, RAND_MAX]
- **ostream & operator << (const onstream &, const NbAleat &) ;** affiche le nombre aléatoire sur le flux passé en argument

b. *Par héritage de la classe précédente*, écrire une classe *NbAleatBorne* qui permet de représenter un nombre entier positif sur un intervalle borné [inf, sup]. Si une méthode n'a pas à être réécrite grâce à l'héritage vous devez le préciser sur la copie. Cette classe doit au moins posséder les méthodes suivantes :

- **NbAleatBorne () ;** construction d'un nombre aléatoire borné dans l'intervalle [0, RAND_MAX]
- **NbAleatBorne (const NbAleatBorne &) ;** construction d'un nombre aléatoire borné par recopie d'un nombre aléatoire borné
- **NbAleatBorne (const NbAleat &) ;** construction d'un nombre aléatoire borné dans l'intervalle [0, RAND_MAX] par recopie d'un nombre aléatoire
- **NbAleatBorne (unsigned int inf, unsigned int sup) ;** construction d'un nombre aléatoire borné dans l'intervalle [inf, sup]
- **virtual ~NbAleat () ;** destruction du nombre aléatoire borné
- **unsigned int Valeur () const ;** retourne la valeur aléatoire bornée générée
- **operator unsigned int () const ;** convertit le nombre aléatoire borné en un entier non signé
- **void Regen () ;** retirage du nombre aléatoire borné dans l'intervalle [inf, sup] fixé dans le constructeur
- **ostream & operator << (const ostream &, const NbAleatBorne &) ;** affiche le nombre aléatoire borné sur le flux passé en argument

Exercice 2 : Les entiers longs positifs

Je vous propose de vous intéresser au stockage des **entiers longs non signés** (positifs !). L'objectif est de concevoir une classe EntiersLongs permettant d'exécuter le programme suivant :

```
#include <conio.h>
#include <iostream.h>

// en-tête des entiers longs
#include "EntiersLongs.h"

void main () {
   EntiersLongs ent1 (68956);
   EntiersLongs ent2 ("678953455671726512345");
   EntiersLongs ent3;
   cout << "ent1 = " << ent1 << endl;
   cout << "ent2 = " << ent2 << endl;
   cout << "ent3 = " << ent3 << endl;

   ent1 << 8; // ajout du chiffre 8 à droite de ent
   ent2 >> 6; // ajout du chiffre 6 à gauche de ent2
   ent3 >> 6; // ajout du chiffre 6 à gauche de ent3
   cout << "ent1 = " << ent1 << endl;
   cout << "ent2 = " << ent2 << endl;
   cout << "ent3 = " << ent3 << endl;

   ent3 = ent1 + ent2; // test de la somme
   cout << "ent3 = " << ent3 << endl;
```

```
   cout << "nb chiffres de ent3 = " << ent3.NbChiffres () << endl;
   cout << "12ème chiffre de ent3 = " << ent3 [11] << endl;

   int val = ent1; // cast ...
   cout << "val = " << val << endl;

   getch ();
}
```

```
U:\MARTIN\ENSEIGNEMENT\MST2\EXAM
ent1 = 689568
ent2 = 6789534556717265123345
ent3 = 0
ent1 = 689568
ent2 = 66789534556717265123345
ent3 = 6
ent3 = 667895345567172728l913
nb chiffres de ent3 = 22
12ème chiffre de ent3 = 7
val = 689568
```

Pour réaliser cette classe, le choix de l'en-tête de cette classe est arrêté. **Il ne peut être modifié que pour ajouter des méthodes.** Voici cet en-tête :

```
#include <iostream.h>

class EntiersLongs
{
      char * m_ch;    // tableau de char pour stocker l'entier long (chaque
                      // chiffre est stocké sous forme d'un caractère)
public:
      // Constructeurs
      EntiersLongs ();
      EntiersLongs (char *);
      EntiersLongs (int);

      EntiersLongs (const EntiersLongs &);   // recopie

      // Destructeur
      virtual ~EntiersLongs ();

      // Fonctions de consultation
      int NbChiffres () const;        // nombre de chiffres du nombre

      // Opérateurs de cast
      operator int () const;
      operator const char * () const;

      // Opérateurs
      EntiersLongs & operator = (const EntiersLongs&);
      int operator [] (int) const; // retourne le i-ème chiffre (0 …)
      EntiersLongs & operator << (int); // ajoute un chiffre à droite
      EntiersLongs & operator >> (int); // ajoute un chiffre à gauche

      // Fonctions amies
      friend EntiersLongs operator + (const EntiersLongs &,
                                      const EntiersLongs &);
      friend EntiersLongs operator * (const EntiersLongs &,
                                      const EntiersLongs &);

      friend ostream & operator << (ostream &, const EntiersLongs &);
};
```

Votre travail consiste à écrire l'implémentation de toutes les méthodes et fonctions qui apparaissent dans cette classe. Attention, la multiplication est complexe.

Exercice 3 : questions

a. Justifiez le mot clef virtual devant le destructeur de la classe EntiersLongs.

b. Le mot-clef static a deux significations en C et une nouvelle signification en C++. Expliquer les trois usages possibles de ce mot-clef (à l'aide d'exemples si nécessaire).

c. Soient les deux classes suivantes:

```
class A                          class B : public A
{                                {
      void f ();                       void f ();
      virtual void g ();               virtual void g ();
      void h ();                       virtual void h ();
      virtual void i ();               void i ();
} ;                              } ;
```

Soit le code suivant :

```
A * a = new A;
A * aa = new B;
B * b = new B;

a-> f ();
a-> g ();
a-> h ();
a-> i ();

aa-> f ();
aa-> g ();
aa-> h ();
aa-> i ();

b-> f ();
b-> g ();
b-> h ();
b-> i ();
```

Précisez pour chaque appel de méthode, quelle méthode est réellement appelée, à savoir A::f, A::g, A::h, A::i, B::f, B::g, B::h ou B::i.