Inhaltsverzeichnis

1. CPP- Static Member

1.1. Aufgabe: Pseudorandom - Ein Zufallszahlengenerator

Es soll ein sogenannter Pseudozufallszahlengenerator erstellt werden, der zur wiederholten Erzeugung von Zahlen im Intervall [0,1] dient. Die in der Regel sehr langen Folgen von generierten Zufallszahlen sollen im Intervall [0,1] möglichst gleichverteilt sein.
Erfahrungen haben gezeigt, dass sich mit der Funktion

$f(x) = (a*x) \bmod n$

mit a= 16807 und n= 2^{31} -1 (= 2147483647) gute Zufallszahlen generieren lassen.

Algorithmus:
a) Man startet mit beliebigem $\mathbf{x_0} \in \{1,...,n-1\}$ (der sogenannten **seed**) und
b) erzeugt gemäß

$x_{i+1} = f(x_i)$ also $x_{i+1} = (a*x_i) \bmod n$

die nächste Zufallszahl.
c) Um die Zufallszahl im Intervall [0,1] zu liefern, dividiert man die Zufallszahl durch n

Hinweise zur Implementierung in C++:

1. Die unsigned long-Konstanten a und n werden als private Membervariablen gespeichert.
2. Die aktuelle Zufallszahl wird als private Membervariable unsigned long currentX gespeichert.
3. Der Defaultkonstruktor initialisiert die Variable currentX auf den Wert 123L.
4. Der allgemeine Konstruktor hat als Parameter mit dem Namen seed den Initialwert für die Variable currentX.
5. Die Methode double nextRand() liefert eine Zufallszahl im Intervall [0,1]. Sie berechnet aus der aktuellen Zufallszahhl - entsprechend der obigen Formel - die neue Zufallszahl, speichert diese in currentX und gibt sie als Zahl im Intervall [0,1] zurück.
6. Die Methode unsigned long nextIntRand(int limit) liefert eine Zufallszahl im Intervall [0,limit[. Sie ruft intern die Methode nextRand() auf und multipliziert diesen mit dem Parameter limit und gibt diesen Wert zurück.

1.1.1. Aufgabe: PseudoRandom

Erstellen Sie die C++-Klasse PseudoRandom (pseudorandom.h und pseudorandom.cpp) nach obigen Vorgaben:

main.cpp

```
…
#inlcude "pseudorandom.h"
int main(){
      int i;

      cout << endl << endl;
      //------------------------------
      //  Würfel
      const unsigned int NO_SIDES= 6;
      int no_rolls= 12000;
      int rollCount[NO_SIDES + 1];
```

```
      long seed;
      for (i=1; i <= NO_SIDES; i++){
            rollCount[i]= 0;
      }

      // prompt for number of rolls
      cout << "Anzahl der Wuerfe: " ;
      cin >> no_rolls;

      // prompt for seed
      cout << "Anfangszahl (Seed) >=1 angeben: " ;
      cin >> seed;

      PseudoRandom rollRnd(seed);
      // würfeln
      for (i=1; i <= no_rolls; i++){
            int index= rollRnd.nextIntRand(NO_SIDES) + 1;
            rollCount[index]++;
      }

      cout << "WURF\tHAEUFIGKEIT\trelative HAEUFIGKEIT"<<endl;
      for (i=1; i <= NO_SIDES; i++){
            cout << i << "\t" << rollCount[i];
            cout << "\t\t" << rollCount[i]/(double)no_rolls << endl;
      }

      cout << endl << endl;
      return 0;
}
```

1.2. Aufgabe: Eine Prüfziffernberechnung

Zur Kennzeichnung von Waren verwendet man den sogenannten EAN-Code (Europäische Artikel-Nummerierung mit 13 Ziffern). Bei Büchern ist die ISBN-Nummer üblich. Bei der letzten Ziffer der Nummer handelt es sich um eine sogenannte Prüfziffer, sodass z.B. einfache Eingabefehler erkannt werden können. Die Prüfziffer berechnet sich aus den übrigen Ziffern.

ISBN-13
Zur Berechnung der Prüfziffer bei der ISBN-13 werden alle zwölf Ziffern der noch unvollständigen ISBN addiert, wobei die Ziffern mit gerader Position (also die 2., 4. usw.) dreifachen Wert erhalten.
Bsp: Eine 5 an 6. Stelle beispielsweise fließt als 15 in die Addition ein.
Von dem Ergebnis dieser Addition wird die letzte Stelle bestimmt, die dann von 10 subtrahiert wird.
Bsp: Also etwa 10 − 4 = 6 bei einem Additionsergebnis von 124.
Dieses Endergebnis ist die Prüfziffer. Ist das Endergebnis indessen 10, ist die Prüfziffer 0.

Formel zur Berechnung der Prüfziffer:

$$z_{13} = 10 - \left(\sum_{i=1}^{n=12} z_i \cdot 3^{(i+1) \bmod 2}\right) \bmod 10$$

$$z_{13} = 10 - \left(\sum_{i=1}^{n=12} z_i \cdot 3^{(i+1) \bmod 2}\right) \bmod 10$$

Das (i+1)mod 2 sorgt für die wechselnde Gewichtung von 1 und 3.

Erstreckt man die Summierung auch auf die Prüfziffer (n = 13), so erhält man bei einer fehlerfreien ISBN als Ergebnis 0.

Beispiel:
978-3-7657-2781-?
Lösung:

9 + 8 + 7 + 5 + 2 + 8 + 3 * (7 + 3 + 6 + 7 + 7 + 1) = 39 + 3 * 31 = 39 + 93 = 132
132 mod 10 = 2
10 − 2 mod 10 = 8 d.h. Die Prüfziffer ist 8

1.2.1. Aufgabe: pruefziffer.cpp

Ersetzen Sie die Fragezeichen mit dem richtigen CPP-Sourcecode.
pruefziffer.cpp

```
class Pruefziffer{
public:
      /**
       * Gibt an, ob die Prüfziffer der ISBN-Nummer gültig ist.
       * verwendet: calcPruefziffer_ISBN(string nummerISBN)
       * @param string nummerISBN: ISBN-Nummer (inkl. '-')
       * @return bool: true, wenn die errechnete Prüfziffer
       *    gleich der letzten Ziffer der ISBN-Nummer ist
       */
(2)   static bool isValid_ISBN(string nummerISBN){
                  ??????????
      }

      /**
       * Berechnet aus der ISBN-13 Nummer die Prüfziffer
       * @param string nummerISBN: ISBN-Nummer (inkl. '-')
       * @return char: errechnete Prüfziffer */
(4)   static char calcPruefziffer_ISBN(string nummerISBN){
            ???????????
            // löscht alle '-' Zeichen
            string nummer= getDigitsOnly(nummerISBN);

            // berechnet die Prüfziffer und gibt diese zurück
            ???????????
      }

private:
      // lokale Hilfsfunktion
      // gibt einen string, der nur aus Ziffern besteht zurück
(2)   static string getDigitsOnly(string nummer){
                  ??????????
      }
};

// ------------------------------------------------------------
int main(){
      string isbn= string("978-3-89771-040-5");
      string isValid;

      cout << "ISBN: " << isbn << endl;
(1)   isValid= ??????????isValid_ISBN(isbn) ? " true" : " false";
      cout << "Pruefziffer ist " << isValid << endl;

      return 0;
}
```

Hinweis:
double std::pow(double, int);
int isdigit(int);