Inlämningsuppgifter

1. Frekvenstabell – stämmer sannolikhetsläran?

Följande program simulerar tärningskast genom att generera slumptal mellan **1** och **6**. Resultatet skrivs ut i tabellform.

```
// Dice.cpp
// Simulerar tärningskast: Slumpar fram tal mellan 1 och 6
// och skriver ut dem i en tabell. Nästlad for-sats
#include <iostream>
using namespace std;

int main()
{
   srand(time(0));              // Skapar variation i slumpen
   int r, k, rader, kolumner;
   cout << "\nAnge antal rader och kolumner (t.ex. 10 15): ";
   cin >> rader >> kolumner;
  cout<<"\nDet blir "<< rader*kolumner<<" tärningskast:\n\n";
   for (r=1; r<=rader; r++)     // Låter en rad att skrivas ut
   {                            // och byter rad.
     for (k=1; k<=kolumner; k++)// Skriver ut den r:te raden.
         cout << 1 + rand() % 6 << "   ";
     cout << '\n';
   }
}
```

Testa programmet **Dice** och vidareutveckla det. Det nya programmet ska undersöka sannolikhetslärans sats om att i idealfallet sannolikheten för ett utfall vid tärningskast är **1/6** och att det praktiska resultatet närmar sig idealfallet, ju större antalet slumpförsök blir. Genomför denna undersökning genom att ställa upp en *frekvenstabell*. Nedan beskrivs frekvenstabellen:

Frekvens är *antalet* förekomster av ett resultat (utfall) bland tärningens **6** möjliga.

Låt programmet genomföra olika antal simuleringar och räkna vid varje simulering frekvensen för varje resultat **1**, ... ,**6** av tärningskastet. T.ex. ska man kunna läsa av från tabellen hur många gånger resultatet **1** förekommer när man kastar tärningen 50 gånger, 100 gånger, 1 000 gånger, 5 000 gånger, 10 000 gånger, osv. Avgör själv hur långt du går. Samma information ska man kunna läsa av från tabellen om tärningskastets andra resultat **2**, ... ,**6**.

Infoga i tabellen även en kolumn som för varje resultat av tärningskastet visar kvoten:

Frekvens / Antalet tärningskast

Denna kvot är den experimentella sannolikheten för ett visst resultat.

Undersök på vilket sätt den experimentella sannolikheten närmar sig den ideala sannolikheten för varje resultat, som enligt sannolikhetsläran borde vara **1/6** eller **0,16667**.

2. Palindrom – en lek med ord

En *palindrom* är en sträng som inte ändras när den läses baklänges. T.ex. är orden *rar*, *död* och *radar* palindromer, även namnet *Hannah* när det stavas så. Men även en text som *ni talar bra latin* är en palindrom om man ignorerar mellanslagen. Och det ska man göra. Därför: vid behandling av sådana texter i ett program låt koden först ta bort alla mellanslag.

Skriv en funktion **`bool palindrom(char *a)`** som avgör om en sträng är en *palindrom* eller ej. Anropa sedan funktionen i **`main()`** efter inmatning av en sträng som ska testas, i ett C++ program som hanterar strängar med pekare. Låt användaren mata in strängar – med eller utan mellanslag – så länge tills man hittat en palindrom. Bjud på möjligheten att avsluta om ingen palindrom hittas och föreslå användaren ett antal palindromer.

3. Collatz algoritmen – rekursiv

I kursen *Programmering med C++* behandlades *Collatz algoritmen* som alltid slutar med 1 oavsett startvärde – ett empiriskt resultat som matematiskt är obevisat:

> Tänk dig ett positivt heltal (startvärde).
> Är talet udda multiplicera det med 3 och addera 1.
> Är talet jämnt dividera det med 2.
> Gör samma sak med resultatet. Fortsätt **tills** du fått 1.

Då implementerade vi Collatz algoritmen *iterativt* med en **`do`**-loop:

```
#include <iostream>
using namespace std;

int main()
{
   int no;
   cout << "\n\tMata in ett pos.heltal:\t";
   cin >> no;
   cout << "\n\t" << no;
```

```
   do
   {
      if (no % 2 == 1)
         no = 3 * no + 1;
      else
         no = no / 2;
         cout << "\n\t" << no;
   } while (no != 1);

   cout << "\n";
}
```

Skriv ett C++ program som implementerar Collatz algoritmen med en rekursiv funktion och anropar den från main().

Ledning: Läs kap ***2.6 Rekursion*** i kursboken, sid 41-44.

Undersök om fenomenet *beräkningskomplexitet* (sid 44) som observerades i Fibonacci rekursionen, även uppträder här. Förklara orsaken. Resonera om för- och nackdelarna av Collatz algoritmens iterativa och rekursiva implementering.