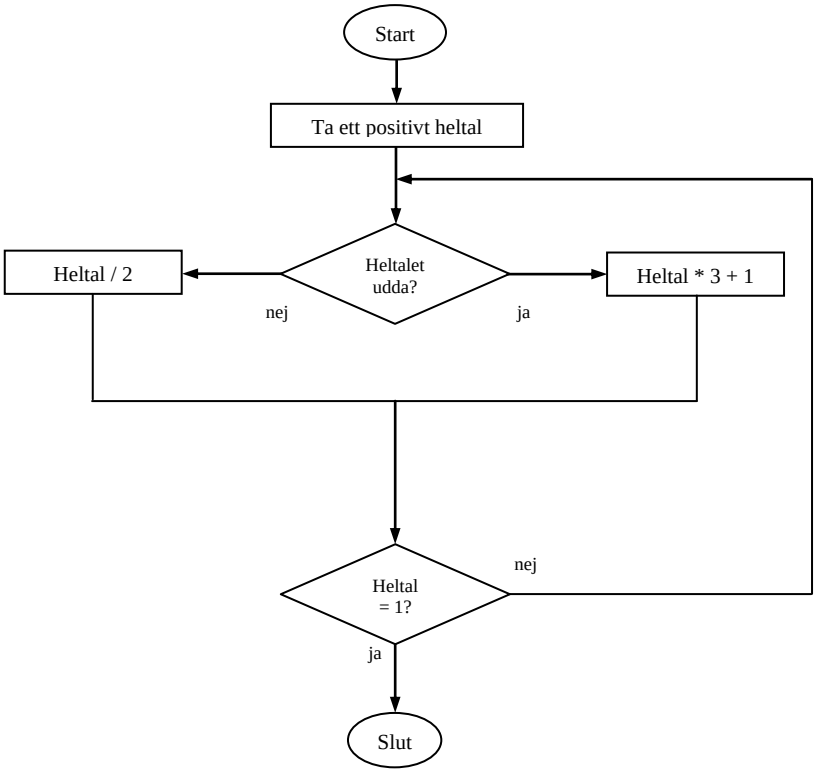
# 1.9 Collatz algoritmen

*Lothar Collatz* (1910-1990) var professor för tillämpad matematik vid Hamburgs Universitet på 60-talet. Som ung student ställde han upp följande uppgift:

> *”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.”*

Det visar sig att talföljderna i denna algoritm, även känd som *Collatz-förmodan* eller *(3n+1)-problemet*, alltid slutar med 1 oavsett startvärde. Dock är detta påstående matematiskt hittills obevisat [*]. Så här ser flödesschemat ut för denna algoritm:

## *Flödesschemat till Collatz algoritmen*




[*] Man kan testa Collatz algoritmen i appen *Mattekollen* där den är kodad i Python. Ladda ned appen eller kör den som Webbapp: **app.mattekollen.se → En mobil pythonmiljö**. Eller kör den direkt som webbapp: beta.mattekollen.se/#/app/coding. Prova koden med olika startvärden för att kolla om algoritmens talföljder alltid slutar med 1.

Flödesschemat visualiserar algoritmens logisla struktur som är grundläggande. Men för att koda kan det vara fördelaktigt att formulera algoritmen även som pseudokod som ligger närmare programkoden än flödesschemat.

## *Pseudokoden till Collatz algoritmen*

```
Läs in ett positivt heltal
SÅ LÄNGE talet ≠ 1 REPETERA:
   OM talet är udda
      multiplicera med 3, addera 1
   ANNARS
      dividera talet med 2
   Skriv ut talet
```

Som man ser har vi redan anpassat texten i pseudokoden till programmering, t.ex. med formuleringar som `Läs in ...` och `Skriv ut ...` . I följande program inplementerar vi Collatz algoritmen:

```
// Collatz.cs
// Läser in ett positivt heltal och tar det gånger 3 + 1 om
// det är udda, annars delar det med 2, tills det blir 1
using System;
class Collatz
{
  static void Main()
  {
    Console.Write("\n\tMata in ett positivt heltal:\t";
    int number = int.Parse(Console.ReadLine());

    Console.Write("\n" + number);  // Startvärde
    while (number != 1)            // Sålänge talet inte är 1
    {
      if (number % 2 == 1)         // Om talet är udda, gångra
         number = 3 * number + 1; // det med 3 och addera 1
      else                        // Om talet är jämnt,
         number = number / 2;     // dela med 2
      Console.Write("\t" + number);
    }
    Console.WriteLine("\n");
  }
}
```

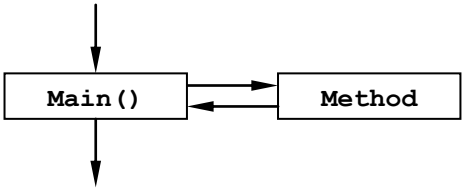
I följande körexempel matas in ett tresiffrigt startvärde. Du kan försöka med andra.

```
        Mata in ett positivt heltal:    135

135     406     203     610     305     916     458     229
688     344     172     86      43      130     65      196

98      49      148     74      37      112     56      28
14      7       22      11      34      17      52      26
13      40      20      10      5       16      8       4
2       1
```

## *Metoder och program i C#*




De flesta känner till begreppet *funktion* från matematiken. Där kan en funktion t.ex. beskrivas med en formel y = f(x) som beräknar ett värde y utgående från ett annat värde x. Även i programmering finns den matematiska synen på funktion som underliggande koncept och historisk utgångspunkt. Men under tiden har den fått en bredare tolkning då den tillämpats på all datoriserad problemlösning.

En *metod* är en funktion som definieras i en *klass*. I objektorienterade programmeringsspråk är metoder inkapslade i klasser. I C# är detta obligatoriskt. Därför finns det i C# till skillnad från C++ inga fristående funktioner. Bortser man från denna överordnade struktur och ser på det ”inifrån”, är funktioner och metoder identiska.

> En metod i C# är en namngiven kodmodul (ett antal satser) i en klass som utförs när metoden anropas. Vid anropet kan den ta emot indata, s.k. parametrar, bearbeta dem och returnera utdata, s.k. returvärde.

Som ”ett antal satser” är en metod en del av en klass som isoleras och skrivs separat som en anropbar modul för att kunna användas även i andra klasser. Man kan jämföra en metod med en ”svart låda” i vilken man stoppar in indata och får ut utdata: Indata kallas även *parametrar* och utdata *returvärde*:


En metod kan ha 0, 1 eller flera parametrar. Den kan ha 0 eller 1 returvärde. En metod kan alltså inte ha flera returvärden. Både parametrarna och returvärdet kan vara tal, tecken, strängar, sanningsvärden eller referenser till objekt. Metoden bearbetar de ev. inkommande parametrarna på ett visst sätt och returnerar ev. ett värde.

Det finns metoder *med* och sådana *utan* returvärde. De senaste kallas för **void**-metoder.

Vi har hittills använt några C#-biblioteksmetoder, t.ex. **Console.Write()**, **Console.WriteLine()**, **Console.Read()**, **Console.ReadLine()**, **int.Parse()**, … utan att behöva veta hur de var kodade, därför: ”svarta lådor”. De var förprogrammerade åt oss och vi använde dem bara för att åstadkomma vissa funktionaliteter. I detta avsnitt ska vi nu lära oss att själva skriva metoder. Men en metod som vi redan har skrivit själva – och det har vi gjort i alla våra programexempel – är metoden **Main()**, för den är obligatorisk. Så här definieras *C# program*:

> Ett C# program är en samling av klasser, av vilka en och endast en måste innehålla metoden **Main()**.
>
> När programmet körs startar exekveringen i **Main()**.

## *Modularisering av Collatz*

Här vill vi modularisera programmet **Collatz** på sid 41. Dvs vi vill separera en del av koden som känns meningsfullt att isolera och skriva den i en metod. Vilken del det ska vara är inte självklart. Nedan ser du ett förslag för ett sådant beslut:

```
// Collatz_mod.cs
// Deklarerar klassen Collatz_mod och definierar i
// den metoden Collatz som utför Collatz algoritmen
using System;

class Collatz_mod
{
   public static void Collatz(int n)  // n formell parameter
   {
      while (n != 1)                  // Så länge talet inte är 1
      {
         if (n % 2 == 1)              // Om talet är udda gångra
            n = 3 * n + 1;            // det med 3 och addera 1
         else                         // Om talet är jämnt,
            n = n / 2;                // dela med 2
         Console.Write("\t" + n);
      }
   }
}
```

Som man ser har vi isolerat algoritmens kärna som utgör själva logiken i det hela, dvs det som Collatz algoritmen väsentligen innehåller och låtit alla andra tekniska detaljerna vara utanför, t.ex. inläsningen av startvärdet, utformningen av utskriften osv. Alla dessa delar stannar kvar i metoden **Main()** som anropar metoden **Collatz()** exakt på samma ställe som koden för Collatz algoritmen stod. Nedan ser vi dessa delar:

```
// Collatz_Test.cs
// Läser in ett positivt heltal number och anropar
// metoden Collatz i klassen Collatz_mod som tar in
// number som parameter och utför Collatz algoritmen
using System;

class Collatz_Test
{
   static void Main()
   {
      Console.Write("\n\tMata in ett positivt heltal:\t");
      int number = int.Parse(Console.ReadLine());

      Console.Write("\n" + number);     // Startvärde
      Collatz_mod.Collatz(number);      // Anrop av metoden
                                        // Collatz med aktuell
                                        // parameter number
      Console.WriteLine("\n");
   }
}
```

Det modulatiserade programmet ovan producerar samma utskrift som det ursprungliga programmet **Collatz** på sid 42. Bara att ”det modulatiserade programmet” till skillnad från tidigare nu består av *två* klasser, lagrade i *två* filer: **Collatz_mod.cs** och **Collatz_Test.cs**. Därför måste båda filerna laddas i samma projekt i Visual Studio när man kör programmet. Annars kan den avgörande satsen

```
Collatz_mod.Collatz(number);
```

dvs anropet av metoden **Collatz()** inte hitta klassen **Collatz_mod** som innehåller metodens definition. Vi har ju separerat den från **Main()**. Den står i en annan klass som i sin tur finns i en annan fil. Båda filer utgör *ett* program och därmed också *ett* projekt i Visual Studio. Det är ju just meningen med modularisering. Nu kan man anropa metoden **Collatz()** även från alla andra program som man ev. skriver, även från sådana som andra skulle skriva. Metoden **Collatz()** har blivit en generell modul som alla utvecklare kan använda sig av.