

Kapitel 4: Loopar

Tillämpning: Bankmedelanden

Denna tillämpning använder loopar för att mata in så många värden som behövs, och som en utvidgning, kontrollerar för giltiga värden som matats in och använder If-satser för att visa ett lämpligt meddelande.

Nästlade Strukturer

- *Nästling* är en programmeringsteknik för att placera en kontrollsats inne i en annan.
- Termen härrör från idén att placera en kartong inne i en annan för att spara plats.
- En programmerare placerar loopar inom loopar, If-strukturer inuti loopar och loopar inuti If-strukturer för att utföra mer komplicerade uppgifter.
- Det är viktigt att placera en komplett struktur helt och hållet inom ett block med en annan för att undvika fel.
- Programlistningen till höger visar en **While**-loop och en **If** struktur innanför en annan **While** loop.
- Observera att en **EndWhile**-sats förekommer två gånger. Datorn "vet" vilken **EndWhile** som hör till vilken **While**.
- Indenteringen (indragningen) i programmet är bara för att göra koden lättläst och logiken tydligare.

Programmet sätter först in en loop som ska fortsätta tills det angivna värdet är noll. Sedan testas det för att se om $a < 0$. Om det är så visar programmet "Måste vara icke-negativt!" och frågar sedan efter ett annat värde för **a**. Men när **a** inte är negativt så beräknas kvadratroten, och en If-sats, och två Disp-satser utförs.

Sammanfattning av de tre looparna:

For(*var, start, slut*) **While** <*villkor*> **Loop**

.....**If** <*villkor*> : **Exit**

EndFor **EndWhile** **EndLoop**

For(används när man "räknar upp" eller processar en aritmetisk sekvens av värden (iteration).

While används när du har möjlighet att helt hoppa över loopkroppen.

Loop används när du är säker på att du vill att loopkroppen ska "köras" åtminstone en gång och den måste innehålla en Exit-sats, vanligtvis som en del av en If-sats.

Syfte:

- Använda räkneverk och ackumulatorsatser.
- Använda en loop i ett program för att få en obestämd mängd av data
- Använda ett "flagg"-värde för att avsluta en loop.

```

perfekt_kvadrat      12/15
Define perfekt_kvadrat()=
Prgm
a:=1
While a>0
Request "Mata in ett tal",a
While a<0
Disp "Måste vara icke negativt!"
Request "Mata in ett positivt tal",a
EndWhile
s:=√a·1.
If s=int(s) Then
Disp "Det är en perfekt kvadrat"
Else
Disp "det är inte en perfekt kvadrat"
EndIf
Disp "Dess kvadratrot är",s
EndWhile
EndPrgm

```

Kapitel 4: Program "Bankmeddelanden"

En bankkund kan ha flera bankkonton på en viss bank. Banken kräver ett minsta genomsnittligt saldo på 1000 kr på alla konton för att undvika att betala serviceavgifter. Om genomsnittet är mellan 1000 kr och 1250 kr skickar banken kunden ett meddelande med en varning om serviceavgift.

Om genomsnittet är mer än 1250 kr skickar banken kunden ett meddelande om att saldot ligger klart över gränsen. Vi ska nu skriva ett program som ger användarinformation om konton. Användaren matar in saldon för sina konton. Programmet kommer att beräkna genomsnittligt saldo och visa antalet konton, genomsnittliga saldovärden och ett meddelande för användaren. Meddelandena kan t.ex. vara: "Serviceavgift tas ut", "Risk för serviceavgift" och "Tack för att du använder våra tjänster"

Vi kan använda två metoder för att mata in ett okänt antal värden:

- Metod 1: fråga efter det totala antalet konton först och använd en **For**-loop för att mata in värdena.
- Metod 2: fråga efter belopp men använd ett s.k. flaggvärde, t.ex. -999, för att indikera att det inte finns fler konton. Denna metod använder en **While**-loop eller en **Repeat** -loop.

I båda metoderna kommer vi att ha ett *löpande* totalvärde för de värden som matats in.

I metod 2 måste vi också räkna *antalet* konton så att vi sedan kan dividera totalvärdet med antalet. Följande information ska då programmet visa:

1) antalet konton, 2) genomsnittligt saldo 3) meddelande till kunden utifrån genomsnittsvärdet på saldona. Se nedan.

Om genomsnittet är under 1000 kr: "Serviceavgift tas ut"

Om genomsnittet är 1000 till 1250 kr: "Risk för serviceavgift"

Om genomsnittet är över 1250 kr: "Tack för att du använder våra tjänster"

Räkneverk och ackumulatorer

En sats som $c:=c+1$ kallas ett räkneverk eftersom den ilägger till 1 till variabeln c varje gång som satsen exekveras.

En sats som $t:=t+n$ kallas en *ackumulator* eftersom den löpande håller reda på totalen för variabeln n . Värdet hos n adderas till variabeln t och sedan lagras den summan tillbaka till variabeln t . I slutet av en loop kommer t att innehålla totalen av n -värdena.

Här är ett exempel som använder ett räkneverk, en ackumulator och ett "flagg-värde" (-999) för att hålla reda på antalet belopp som matats in i programmet.

Prgm	Kommentarer
Local <i>räkneverk, belopp, totalt</i>	initiera variabler
<i>totalt:=0</i>	
<i>räkneverk:=1</i>	
Request "Belopp?", <i>belopp</i>	första beloppet
While <i>belopp</i> ≠-999	så länge det inte är -999
<i>räkneverk:=räkneverk+1</i>	addera 1 till räkneverket
<i>totalt:=totalt+belopp</i>	lägg till belopp till totalt
Request "Belopp?", <i>belopp</i>	fråga efter nytt belopp
EndWhile	
Disp "Antal=", <i>räkneverk</i>	
Disp "Totalt=", <i>totalt</i>	
EndPrgm	

```

räkneverk_ackum()
Belopp? 400
Belopp? 1200
Belopp? 900
Belopp? 1000
Belopp? -999
Antal= 5
Totalt= 3500
Klar

```

While loopen ovan fortsätter räkna och ackumulera belopp så länge -999 inte matas in. När -999 matas in visas stoppar Loopen och resultaten visas.

Utvidgning

Som en del av inmatningsrutinen ska du se till att värden som inmatas är giltiga (större än 0). Om inmatat värde ligger utanför detta intervall ska lämplig åtgärd vidtas.