Lektion 3

Del 1 kapitel 8, Kontrollstrukturer

Inlärningsmål

- Att kunna förstå grundläggande problemlösnings tekniker
- Att kunna utveckla algoritmer för stegvis topdown förfining av ett problem
- Att kunna använda if och if/else uttryck
- Att kunna använda whie loopar
- Att förstå räknarstyrd exekvering av ett program
- Att kunna använda ”increment, decrement” och tilldelnings operatorer

Introduktion

Grundläggande för all problemlösning i program är möjligheten att använda kontrollstrukturer.

Algoritmer

All problemlösning med dator kan brytas ner i en serie beräkningar och val i en viss ordning. Detta kallas en algoritm. Algoritmer kan vara olika effektiva och precisa i sin lösning.

Kontrollstrukturer

JavaScript har ett antal olika funktioner för program styrning. Den första av dessa, if-satsen har vi redan gått in lite på. Denna styr exekveringen utifrån ett villkor som kan var sant eller falskt. Andra kontrollstrukturer är while loopen och for loopen där den första styrs av ett villkor och den andra av en räknare.

If

En enkel if satts ser ut som följer

```
if(nummer >= 10)
        writeln(”Numret är större än eller lika med 10”);
```

Villkoret i detta fallet är nummer >= 10. Om detta evalueras till sant utföres raden som följer annars hoppas den över. Vi kan lägga till en **else** satts till detta och den kommer att utföras om villkoret evalueras till false.

```
if(nummer >= 10)
        writeln(”Numret är större än eller lika med 10”);
else
        writeln(”Numret är mindre än 10”);
```

En if satts kan även nätlas, alltså en if satts kan innehålla ytterligare en if satts.

```
if(nummer >= 10)
        writeln(”Numret är större än eller lika med 10”);
else
        if(nummer < 5)
                writeln(”Numret är mindre än 5”);
        else
                writeln(”Numret är mindre än 10 men större än 4”);
```

En nästlad if satts kan även skrivas enligt följande.

```
if(nummer >= 10)
        writeln(”Numret är större än eller lika med 10”);
else if(nummer < 5)
        writeln(”Numret är mindre än 5”);
        else
        writeln(”Numret är mindre än 10 men större än 4”);
```

Ett annat sätt att uforma vad som kan liknas med en if satts är att använda notationen ? : enligt följande

```
document.write(nummer  > 10 ? ”Större än 10” : ”Mindre än 11”);
```

Om nummer är större än 10 skriv Större än 10 annars skriv Mindre än 11.

While loopar

Med en while loop kan man konstruera en programsekvens som skall utföras om och om igen intill ett bestämt villkor är uppfyllt. En while satts deklareras enligt följande.

```
While (villkor)
        Utför detta;
```

Och med värden insatta så här

```
While (produkt <= 1000 )
        produkt = 2 * produkt;
```

En algorittm för en while satts kan se enlig följande *så länge produkt är mindre än eller lika med 100, dubbla produkten.*

I figur 8.7 använder man en while loop för att mata in 10 värden och sedan räkna ut medelvärde. På rad 25 testar vi värdet på gradeCounter, så länge det är mindre än 10 fortsätter loopen. På rad 37 räknar vi upp gradeCounter med 1 för varje varv.

Formulera algoritmer enligt uppifrån och ner med stegvis förfining

I vårt förra exempel var antalet varv känt i förväg. Nu skall vi studera ett exempel där vi i förväg inte vet hur många varv vi behöver köra programmet. Ett sätt att göra detta är att låta användaren ”flagga” när loopen skall avbrytas. Detta gör man genom att mata in ett i förväg bestämt specialvärde i inmatningsfältet. Det är viktigt att avbrottsvillkor väljs så att det inte kan överensstämma med något värde som ingår i den serie som man vill mata in. I exeplet i boken har man valt att använda värdet –1

som avbrottsvillkor. Här använder man ”top-down, stepwise refinement”. Man delar in problemet i mindre steg.
initiera total till noll
initiera gradeCounter till noll

Mata I det första värdet (möjligen avbrottsvärdet)
Så lännge användaren inte har matat in avbrottsvärdet
Lägg till värdet till total
Lägg till ett till grade counter
Mata in nästa värde (Möjligen avbrottsvärdet)

Om räknaren är skild från noll
Sätt medel till total delad med räknaren
Skriv ut medelvärdet
Annars
Skriv “Inga värden inmatade”
Det är viktigt att se till att testa på möjligheten att dela med noll vilket skulle ge ett fel. Det är oftast utvecklingen av en lämplig algoritm som är den svåraste delen i arbetet med att ta fram ett program. Programmet demonstreras i figur 8.9.

Formulera algoritmer enligt uppifrån och ner med stegvis förfining med nästlande kontrollstrukturer

Problemet liknar det i föregående exempel men här bokför vi endast om en person är godkänd eller underkänd. Dess utom vill vi om fler än åtta är godkända skriva ut ett meddelande om att kraven bör höjas. Här följer pevsdokoden för uppgiften.
initiera passes till noll
initiera failures till noll
initiera student till ett
Så länge student counter är mindre än eller lika med tio
Mata in nästa resultat
Om studenten är godkänd
lägg till ett till passes
Annars
lägg till ett till failures
lägg till ett till student counter
Skriv ut antalet godkända
Skriv ut antalet failures
Om mer än tio godkända
Skriv ut “Raise tuition”
Implementationen visas i figur 8.11.

Öknings och minsknings operatorer

Det finns precis som i Java och c++ ett antal operatorer för att räkna upp och ned variabler.

Operator	Namn	exempel	Förklaring
++	preincrement	**++a**	Öka värdet på a med 1, använd sedan det nya värdet på **a** i uttrycket som innefattar **a.**
++	postincrement	**a++**	Använd det nuvarande värdet på **a** I uttrycket och öka sedan med ett
--	predecrement	**--b**	Miska värdet på a med 1, använd sedan det nya värdet på **a** i uttrycket som innefattar **a.**
--	postdecrement	**b--**	Använd det nuvarande värdet på **a** I uttrycket och minska sedan med ett

Dessa operatorer visas i figur 8.14

Lektion 3

Del 2, Kapitel 9, Mera kontrollstrukturer

Inlärningsmål

- Att kunna använda kontrollstrukturerna **for** och **while** för att styr program exekveringen.
- Att förstå och kunna använde kontrollstrukturen **switch**
- Att kunna använda **break** och **continue**

Introduktion

Kontrollstrukturer är något som är grundläggande för all program konstruktion oavsett språk. De är dessutom oftast mycket lika varandra i utformning och syntax. Vi fortsätter i denna del den genomgång vi påbörjade i föregående del.

Räknarstyrda repeterande loopar

While loopen

Den första typen av loop som vi skall titta närmare på är whileloopen. En whileloop styrs av ett avbrottsvillkor, så länge som villkoret är sant fortsätter repetitionen av loopen. Syntax för en whileloop ser ut som följer.

```
While (testvariabel villkor testvärde)
{
       kod som skall exekveras;
}
```

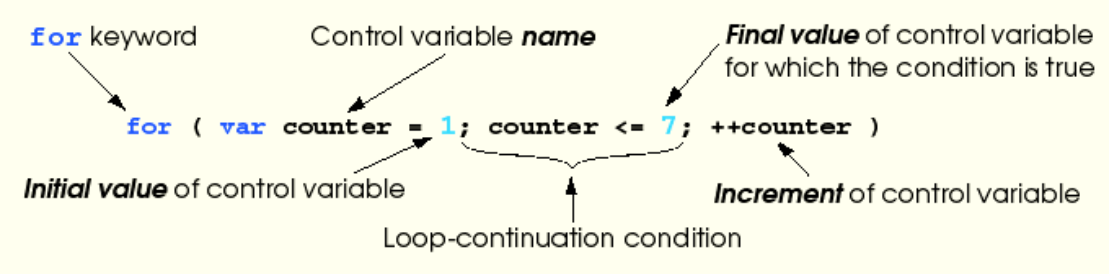
Se på exempel/figur 9.1. På rad 16 i koden inleds whileloopen med nyckelordet while följt av en villkorssatts. Villkoret säger; så länge counter är mindre eller lika med 7 så fortsätt loopa. På raderna 17 till 19 finns den kod som skall exekveras och på rad 20 räknas vår kontrollvariabel upp med ett för varje repetition.

Just detta exempel skriver ut en kort text där fontstorleken räknas upp för varje varv. Fontens storlek anges här i ex vilket anger storleken på texten relativt storleken på lilla x för normalteckensnittet.

For loopen

For loopar är den bekvämaste loopen för räknarstyrda loopar. En for loop ombesörjer själv uppräkning av test variabeln och des syntax ser ut enligt följande.
For (var uppräkningsvariabel; uppräkningsvariabel villkor testvärde; uppräkning)

```
{
    Kod som skall exikveras;
}
```

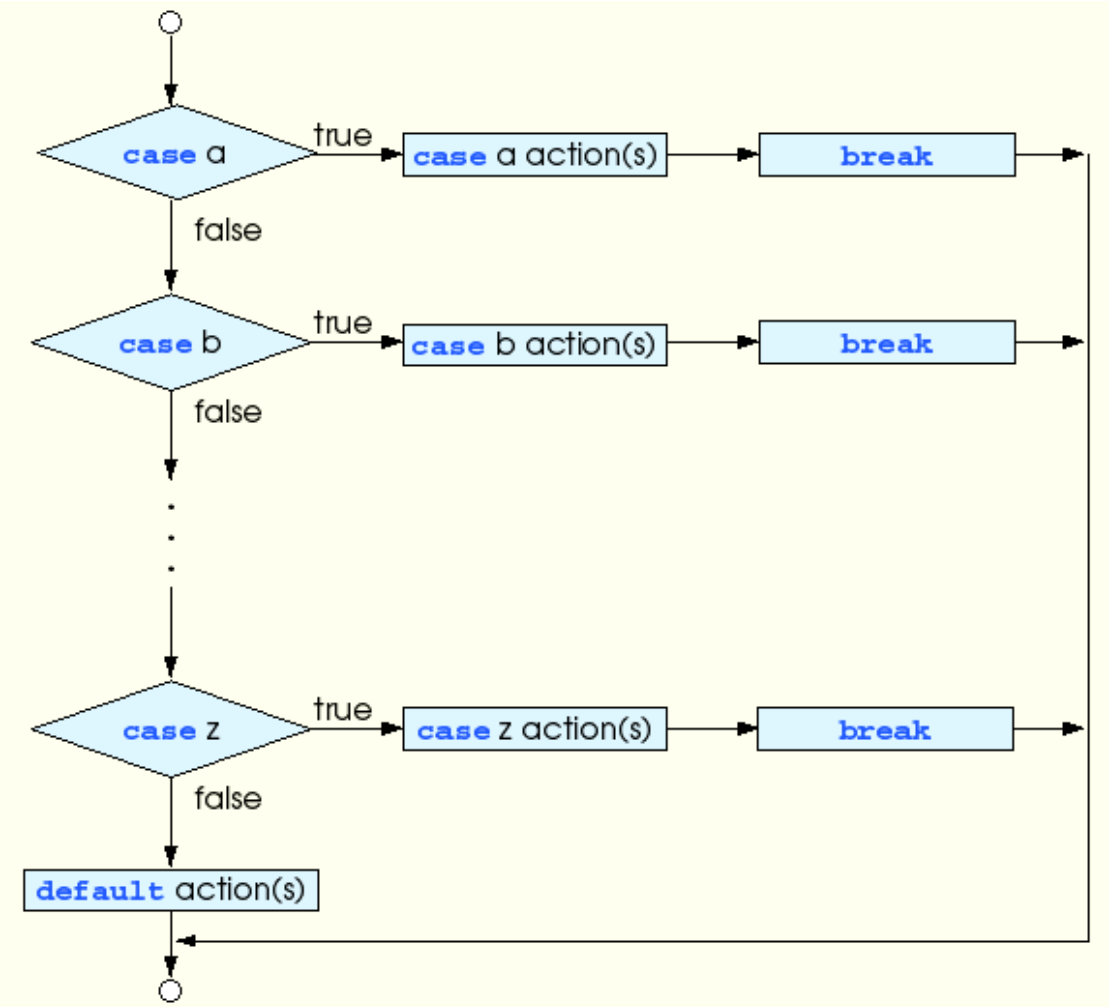


Se exempel/fig 9.2. På rad 17 börjar forlopen med nyckelordet for, där efter inleds parentesen med att uppräkningsvariabeln skapas och initieras till 1. Där efter följer ett semikolon som används för att skilja de olika delarna i parentesen åt. Nästa del är själva avbrottsvillkoret som i detta fallet säger; så länge counter är mindre eller lika med 7, fortsätt. Efter nästa semikolon kommer uppräkningssatsen, ++counter. Den räknar upp counter med ett för varje varv. Detta exempel utför det samma som det föregående.

Exempel/fig. 9.6 visar hur man dynamiskt med hjälp av en forloop bygger upp en tabell med värden. På rad 16 till 24 börjar vi bygga upp den sträng som bildar xhtml koden för tabellen och formaterar des utseende. I forloopen på raderna 26 till 31 lägger vi till rader och fyller den med värden. En ny rad läggs till för varje varav som forloopen snurrar.

Switch Case satsen

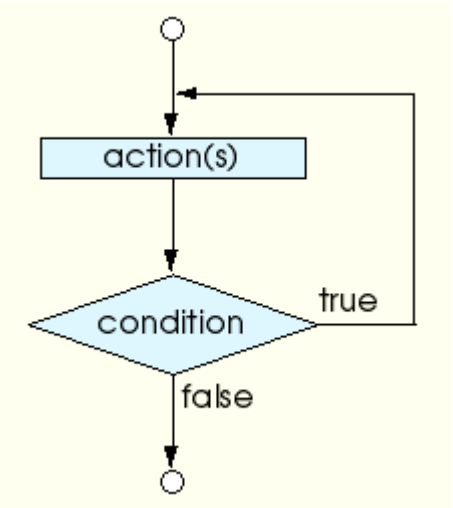
En annan mycket användbar kontrollstruktur är switch case satsen. Denna används för att testa på ett antal möjliga val och därefter utföra lämpliga instruktioner beroende på val. Detta visas i exempel/fig. 9.7. Vi börjar med att på rad 20, 21 hämta ett värde som läggs i variabeln choice med hjälp av en inmatningsprompt. Denna variabel skall användas för att avgöra vilket val vi skall göra. På rad 20 inleds switch satsen. Inom parentes står vår variabel som skall användas för att göra ett val. Sedan följer på raderna 23, 29, 34 en case sats för varje giltigt val. På rad 39 kommer en default sats som exekveras om inget eller fel värde angetts. Prova att köra detta exempel och studera koden.



Logisk struktur för Switch

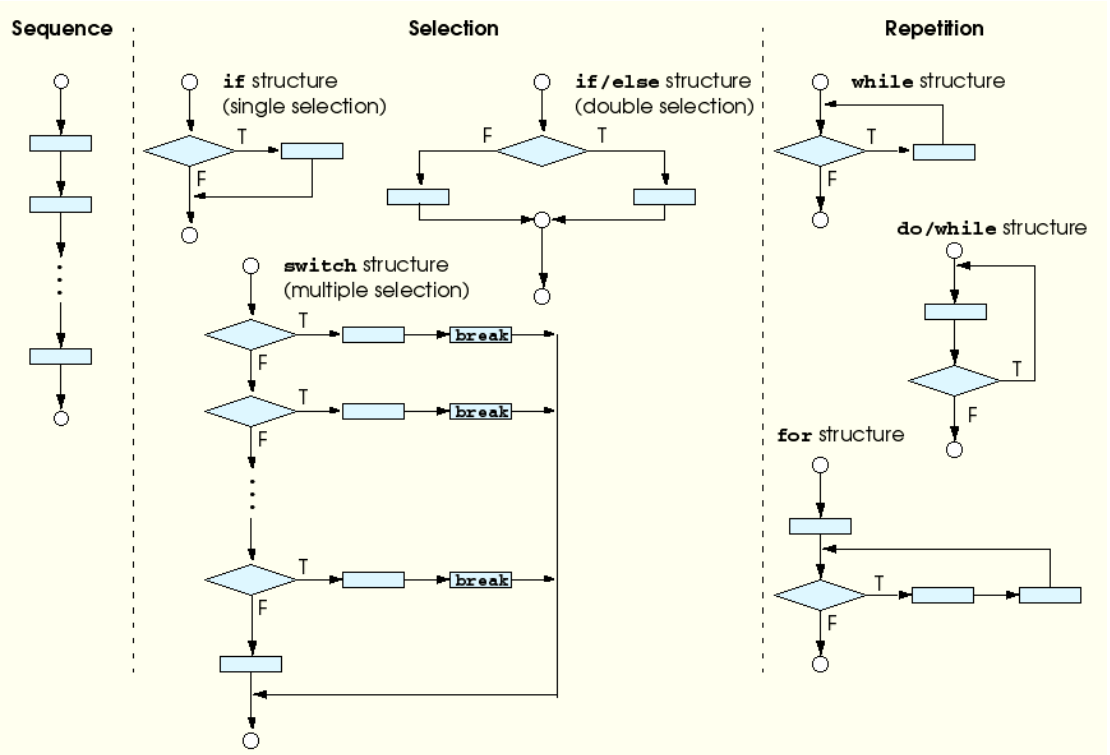
Do while

While satsen finns även i en do while variant. En sådan struktur exekveras alltid minst en gång då avbrottsvillkoret kommer sist i strukturen. I exempel/fig.9.9 används do/while strukturen. På rad 16 kommer do satsen som inleder strukturen, därefter kommer den kod som skall exekveras och sist på rad 22 kommer while med avbrottsvillkoret.



Break och Contiue

Med Break och continiue satserna kan man styra villkorliga avbrott med t.ex. If satser i en for eller while sats. EX: if (count ==5) break tvingar en loop där uttrycket finns att avbrytas i förtid. Bägge sattserna kan kopplas till en label som kan plaseras i koden någon stans. En label skriv t.ex. så här: stop: och för att anropa denna skrivs t.ex. Med en break satts enligt följande break stopp;



Sammanställning av del logiska strukturen hos kontrollstrukturer.