Programmering i PHP

Marcus Rejås

marcus@rejas.se

Magnus Määttä

magnus@php.net

Programmering i PHP

av Marcus Rejås och Magnus Määttä

Publicerad Utkast

Copyright © 2003-2005 Marcus Rejås och Magnus Määttä

Denna bok är anpassad för gymnasieskolans kurser Programmering A och B med kurskoderna DTR1207 och DTR1208 (PPHP1408). Den kan naturligtvis användas även i andra sammanhang, till exempel självstudier, studiecirklar eller annan lärarledd utbildning.

Till boken kommer att finnas förslag på övningar man kan utföra för att praktiskt utöva de färdigheter man skaffar sig under studierna.

Var och en äger rätt att kopiera, distribuera och/eller modifiera detta dokument under villkoren i licensen "GNU Free Documentation License", version 1.2 eller senare publicerad av Free Software Foundation, med de invarianta avsnitten Appendix A och Appendix B, utan framsidestexter och utan baksidestexter. En kopia av denna licens finns med i avsnittet med titeln "GNU Free Documentation License".
Det vill säga, du kan fritt ladda ner, vidaredistribuera och kopiera denna bok. Du får ändra den om du vill (se licenstexten). Tryckta böcker kan köpas av TriNix AB i Helsingborg, telefon 042-127800.

Revisionshistorik
Revision $Id: programmeringAB.xml,v 1.23 2006/06/11 10:27:10 rejas Exp $
*** Utvecklingsversion ***

Innehållsförteckning

Tabellförteckning

Förord

1. Tack till

Vi vill börja denna bok med att tacka alla som hjälpt oss med den. Speciellt vill vi tacka Micke Karlsson som föreslagit förbättringar, Jerry Segerholm som hittat stavfel och Daniel Wahlgren som hittat både stavfel och sakfel. Tack!

Vi vill också tacka dig som läser denna bok. Hittar du något som är fel eller som du tycker att vi kan göra på något bättre sätt så tveka inte att höra av dig till oss. Våra adresser står i början av boken. Vi har våra begränsningar som författare men genom att släppa denna bok fri förväntar vi oss att du som läsare skall skicka kommentarer så att vi tillsammans kan göra den mycket bättre än vad vi och kanske du skulle ensamma.

Kapitel 1. Kort historik

1.1. Före 1900

Man började faktiskt att utveckla den metodiken som används i dagens datorer redan på 1800-talet. Den första kalkylatorn som räknade digitalt uppfann och byggde engelsmannen Charles Babbage 1832! Den opererade på sexsiffriga nummer och kunde lösa andragradspolynom med sex siffrors noggrannhet. Efter denna maskin började Babbage på en ännu större maskin, the analytical engine. Denna maskin var en enorm uppfinning som introducerade saker som vi fortfarande känner igen i dagens datorer. Maskinen hade in- och utmatningsenhet, minne samt kalkylator och kontrollenhet (processor). Dessvärre fick han den aldrig att fungera. Idag vet vi varför, det var en enorm uppgift som inte gick att lösa med dåtidens kunskaper och verktyg. Babbage dog i 1871. Då var det inte många som visste vem han var men nu nämns han i varenda kurs som innefattar datorhistoria världen över.

Mycket mer saker händer under 1800-talet. Men vi hoppar direkt till 1900-talet.

1.2. 1900-talet

1936 skrev matematikern Alan Turing en rapport som bland annat beskriver hur en digital dator skulle fungera. Rapporten beskriver en matematisk maskin som kunde utföra logiska operationer och läsa, skriva och radera binära symboler (1:or och 0:or) på en oändlig remsa. Dessa kom sedan att kallas Turing-maskiner.

1941 blev den första fungerande turingmaskinen klar. Den skapades av Konrad Zuse. Den var fritt programmerbar och helautomatisk. Den hade en klockfrekvens på 5.33 Hz och vad byggd med reläer. Det dröjde några år till innan datorerna blev elektroniska.

Den första moderna datorn blev en maskin som hette ENIAC. Den stod klar 1946. ENIAC är en förkorting för Electronic Numerical Integrator And Computer. Den var stor som ett hus (18000 elektronrör) och kunde mindre än den enklaste kalkylator kan idag. En som var inblandad i utvecklingen av ENIAC var John von Neumann. Von Neumann skapade en datorarkitektur, von Neumann-arkitekturen, som datorer än idag byggs efter. Det var den första arkitekturen där beräkningsenhet och minne var separerade. Mellan dem fanns det en buss över vilken data och instruktioner transporterades. Von Neumann anses av många vara den moderna datorns fader. Den första Von Neuman-datorn, som han själv var med och byggde, hette EDVAC.

1.3. Nutid

Utvecklingen står på intet sätt still. Datorerna utvecklas ständigt. Varje år blir datorerna snabbare och

billigare. Det lär dock dröja ett tag innan något så revolutionerande som Babbages kalkylator eller ENIAC ser dagens ljus. Sen hur lång tid det tar, det återstår att se.

1.4. Mer läsning

På adressen *http://www2.fht-esslingen.de/studentisches/Computer_Geschichte/fold1.html* (http://www2.fht-esslingen.de/studentisches/Computer_Geschichte/fold1.html) finns en bra tidslinje för datorns historia.

Kapitel 2. Programmeringsspråk

Detta avsnitt ger en orientering om olika programmeringsspråk och varför det finns så många och några grundläggande egenskaper hos olika språk och olika familjer av språk.

2.1. Olika språk till olika saker

Det finns idag hundratals olika programmeringsspråk. De har kommit till av olika anledningar och har olika syften. Vissa är besläktade och har arv från varandra och andra har utvecklats för att fylla ett speciellt ändamål.

Anledningen till att det finns så många språk och alltid kommer nya är naturligtvis att de är bra på olika saker och att olika personer uppskattar språken olika mycket.

Vi kan börja med att dela upp språken i två grupper efter hur programmen kompileras och körs. Alla program skrivs i någon form av källkod. Denna källkod måste sedan översättas till något som en dator kan förstå. Datorn förstår bara 1 och 0, på eller av. Detta kallas kompilering och kan göras vid olika tillfällen.

2.2. Kompilerande språk

Datorn kan ju inte läsa utan det som man skriver i sina program måste översättas till något som datorn förstår. Det datorn förstår kallas för maskinkod eller binärkod. Denna kod består av maskininstruktioner som är, näst intill, omöjliga för en människa att förstå.

Ett kompilerande språk är ett programmeringsspråk där källkoden med hjälp av olika verktyg översätts till maskinkod. Maskinkoden blir då ett fristående program som kan köras direkt av datorn. Man säger att man kompilerar koden. Koden kopileras alltså i samband med utvecklingen och inte i samband med exekveringen av programmet. Mer om det i ett annat avsnitt.

Exempel på kompilerande språk är: C, C++, Pascal och många fler.

2.3. Interpreterande språk

Ett interpreterande programspråk, som också kallas skriptspråk, är ett programspråk som inte kompilerar det program som programmeraren har skrivit förrän det körs, och som gör det varje gång det körs. Ibland

kompileras hela programmet innan det körs och ibland kompileras det rad för rad. Detta gör att utvecklingen går snabbare eftersom programmet inte behöver kompileras vid testkörning men det innebår också att det färdiga programmet blir långsammare eftersom det måste kompileras varje gång det körs.

Motorn som kör ett interpreterande språk kallas interpretator, programtolk eller tolkare.

Exempel på interpreterande språk är: Perl, PHP, Python, TCL, Bash.

2.4. Andra typer av språk

Det finns andra sätt att lösa uppgiften på. Till exempel att man kompilerar koden till en mellankod som sedan en interpreterare tolkar. Så fungerar till exempel Java.

2.5. För- och nackdelar

Tabell 2-1. Fördeler och nackdelar med kompilerande och interpreterande språk.

Fördelar	**Nackdelar**
Kompilerande	
Snabbt att köra	Långsam programmering
Lätt att distribuera	Svårt att portera
Interpreterande	
Snabb programmering	Långsamt att köra
Mycket lätt att portera (om tolkare finns)	Svårare att distribuera (tolkare måste finnas)

2.6. Exempel på olika språk som ni bör känna till

Det finns tusentals, jo säkert, olika programmeringsspråk. Här listas de jag tycker ni bör känna till och hur programmet "Hello World!" ser ut i dem.

2.6.1. C

C är ett av de mest utbredda språken. Det är en vidareutveckling av språket B. Utvecklades vid AT&T Bell Labs samtidigt med operativsystemet UNIX®. C anses som ett ganska maskinnära språk. Flera operativsystem är till stor det skrivna i C. Hello World i C ser ut så här:

Exempel 2-1. Hello World i C

```
/*
 * Hello World i C
 */
#include <stdio.h>

int main() {
        printf("Hello, world!\n");
        return 0;
}
```

2.6.2. C++

C++ utveckladen av Bjarne Stroustrup vid AT&T Bell Labs och är en utökning av programspråket C. Till skillnad från C så är C++ objektorienterat (eller har bättre stöd för det i alla fall).

Hello World i C++ ser ut så här:

Exempel 2-2. Hello World i C++

```
/*
 * Hello World i C++
 */
#include <iostream>

int main () {
        cout << "Hello world" << endl;
}
```

2.6.3. C# (C-sharp eller Ciss)

Objektorienterat språk utvecklat av Microsoft för att möta Java från Sun. Språket är, liksom Java, halvinterpreterande och har mycket influenser från C och C++.

Hello World i C# ser ut så här:

Exempel 2-3. Hello World i C#

```
//
// Hello World i C#
//
using System;
class HelloWorld {
        static void Main() {
                Console.WriteLine("Hello, world!");
        }
}
```

2.6.4. Java

Objektorienterat språk utvecklat av Sun Microsystems. Språket är halvinterpreterande och har mycket influenser från C och C++. En stor fördel med Java är att program skrivna i Java är nästan platformsoberoende. Du kan alltså köra dem i alla miljöer till vilka det finns en javamotor.

Hello World i Java ser ut så här:

Exempel 2-4. Hello World i Java

```
//
// Hello World i Java
//
class HelloWorld {
        public static void main (String s[]) {
                System.out.println("Hello world");
        }
}
```

2.6.5. Mer läsning

En sida med jättemånga exempel på "Hello World!" i olika språk. (*http://www2.latech.edu/~acm/HelloWorld.shtml* (http://www2.latech.edu/~acm/HelloWorld.shtml))

Kapitel 3. Från källkod till program

I detta avsnitt beskrivs vad en kompilator är och varför den används. Detta ingår i kursen men används inte i PHP eftersom det är ett så kallat skriptspråk. I fallet med PHP tar en tolkare hand om allt detta på servern när programmet körs.

3.1. Kompilering

När man "kompilerar" ett program så utför man egentligen 3 steg. Ett fjärde steg utförs precis innan programmet laddas för att köras. Dessa steg behandlar vi här.

Stegen är förbehandling av källkoden, kompilering, assemblering och slutligen länkning. Dessa steg behandlas här.

3.1.1. Förbehandling av källkoden (preprocessing)

Det första som händer med din källkod är att den går igenom något som kallas en preprocessor. Den tar bort alla kommentarer, som ju i alla fall bara betyder något för människor och formaterar din kod så att den passar kompilatorn.

3.1.2. Kompilering

Sedan är det kompilatorns tur. Det översätter källkoden till ett mellanspråk anpassat för den målmiljö som man kompilerar för. Detta mellanspråk kallas assembler och skall sedan behandlas av en assemblerare.

3.1.3. Assemblering

Assemleraren översätter assemblerkoden till maskinkod. Denna kod är relokerbar, det vill säga den är inte bunden till fasta adresser i minnet. Dessa relokerbara adresser måste dock ändras innan programmet kan köras, det gör laddaren (loader).

3.1.4. Laddning och länkning

Detta är egentligen två steg men de görs nästan alltid tillsammans och de görs varje gång programmet skall köras. De görs oftast av samma rutin. Laddning innebär att programmet flyttas till ett ställe i minnet där det kan köras och se till att adresserna i programmet anpassas efter det ställe det skall köras. Länkaren behöver bara jobba om det körbara programmet behöver delar från flera olika filer, så kallade delade bibliotek (shared libraries, dll:er). Länkaren ser då till att hänvisningarna till dessa i programmet blir riktiga och kontrollerar att de finns tillgängliga.

3.1.5. Mer läsning

Om du är intresserad kan du läsa denna länk
http://techpubs.sgi.com/library/dynaweb_docs/0620/SGI_Developer/books/Pascal_PG/sgi_html/ch02.html
(http://techpubs.sgi.com/library/dynaweb_docs/0620/SGI_Developer/books/Pascal_PG/sgi_html/ch02.html) där kompileringen i Pascal beskrivs mer ingående.

Kapitel 4. Hallå Världen!

Detta avsnitt beskriver hur ett enkelt PHP-script är uppbyggt.

4.1. Programmeringsmiljön

Som vi har sagt tidigare så skapar man ett program på följande sätt (något förenklat):

- Skapa PHP-fil
- Kopiera till servern
- Provkör

Du kan använda vilken editor du vill för att skapa php-filer. Notepad fungerar alldeles utmärkt men det finns de med fler finesser. Vim är en av dem. Den har stöd bland annat för färgläggning av kod och finns för både Linux® och Windows att hämta från deras hemsida: *http://vim.sf.net* (http://vim.sf.net)

4.2. Hallå världen!

Nästan alla programmeringskurser börjar med att man skapar ett program som heter Hello World. Hello World är ett program som inte gör något annat än att skriva ut "Hello World" på skärmen.

Vi börjar med att titta på hur det ser ut i PHP.

Exempel 4-1. Hello World i PHP

```
<html>
<head>
   <title>Hallå Världen!</title>
</head>
<body>
   <p>
      <?php
         echo "Hallå Världen!";
      ?>
   </p>
</body>
</html>
```

Så, vad gör nu detta program? Som ni säkert kan se så är det HTML i början och slutet av filen. Det enda av filen som är PHP är mellan tecknen <?php och ?>. Det är för att tolkaren skall veta när den skall exekvera raderna istället för att skriva ut dem.

Den enda PHP-koden i detta lilla program är echo "Hallå Världen!";. Echo talar om att något skall skrivas ut (Hallå Världen!) och semikolonet talar om att en sats i programmet slutar. En sats kan omfatta flera rader men avslutas alltid av ett semikolon.

Man kan självklart skriva ett "Hello World!" program som enbart använder sig av PHP och ingen HTML (dock utan den formatering som kommer med HTML).

Exempel 4-2. Hello World i PHP utan HTML

```
<?php
echo "Hallå Världen!";
?>
```

4.3. Övningsuppgifter

- Ändra programmet så att det skriver ut något annat än "Hallå Världen!". Testa att det fungerar som du vill.
- Gör så att två rader skrivs ut. (Använd HTML-koder).

Kapitel 5. Webbrowser, webbserver och program

När man skriver och kör ett PHP-program är det många delar som skall samverka. Man skriver ett program som skall tolkas av en PHP-tolk. Sedan skall utdata av detta program hanteras av en webbläsare. Hur detta hänger ihop förklaras i detta kapitel.

5.1. Webbläsaren

Webbläsaren är den del av kedjan som är närmast användaren. Webbläsaren är det som du eller dina användare kommer i kontakt med. För att webbläsaren skall visa vettiga saker för användaren gäller att den matas med vettiga saker från ditt program.

Webbläsaren förstår HTML [1] så det som skall komma till webbläsaren är HTML. Webbläsaren förstår inte PHP-kod så PHP-koden måste tolkas av en PHP-tolk på webbservern.

5.2. Webbservern

Webbservern har till uppgift att serva webbläsaren med webbsidor. Kör man PHP så är det även på webbservern som PHP-koden tolkas. PHP-tolken gör inget för att det som kommer ut ur PHP-koden skall bli giltig HTML utan det är upp till dig som programmerare att se till att den är det.

Webbservern har ofta en mängd olika funktioner beroende på vad den för tillfället skall göra. Du kan till exempel ha rena HTML-sidor parallellt med din PHP-filer. Webbservern vet vilka sidor som är PHP och vilka som är HTML och hanterar dem därefter.

5.3. Program

Programmet, eller skriptet, skall fungera så att det skriver ut HTML-kod som webbservern kan skicka till användarens webbläsare.

Noter

1. Moderna webbläsare förstår en massa annat också, men det som är relevant i denna kurs är HTML

Kapitel 6. Variabler

Detta avsnitt beskriver vad variabler är och hur man använder dem i ett PHP-script. Det tar även upp skillnaden mellan variabler i PHP och i andra vanliga kompilerande språk.

6.1. Vad är en variabel

En variabel är en platshållare för ett värde som ändras under programmets gång. Motsatsen till variabel är en konstant. En konstant kan eller får inte ändras under programmets gång.

Ett exempel på en variabel kan vara termer i en addition. Tänk dig att du vill göra ett program som skall ränka ut en summa. Man skulle kunna göra programmet med bara konstanter, det skulle se ut så här:

Exempel 6-1. Exempel med konstanter

```
<html>
<head>
    <title>Addition</title>
</head>
<body>
    <?php
       echo "Summan av talen 1 och 2 är: ", 1 + 2;
    ?>
</body>
</html>
```

Som du kan se så räknar PHP ut vad 1 + 2 blir. Men vad har man för nytta av ett program som bara räknar ut 1 + 2. Vi vill ju kunna mata in vilka värden som helst för termerna.

Se nedanstående exempel:

Exempel 6-2. Exempel med variabler

```
<html>
<head>
<title>Addition</title>
</head>
<body>
<?php
$tal1 = 2;
$tal2 = 3;
```

```
echo "Summan av talen $tal1 och $tal2 är: ", $tal1 + $tal2;
?>
</body>
</html>
```

I exemplet ovan använder vi två variabler, $tal1 och $tal2. I PHP börjar alla variabelnamn med ett dollartecken. Programmet ovan är dock lika statiskt som det fösta programmet. Våra variabler får ju samma värde (2 och 3) varje gång programmet körs. De kan inte påverkas av något.

Vi tar ytterligare ett exempel:

Exempel 6-3. Exempel med variabler och $_GET

```
<html>
<head>
<title>Addition</title>
</head>
<body>
<?php
$tal1 = $_GET['tal1'];
$tal2 = $_GET['tal2'];
echo "Summan av talen $tal1 och $tal2 är: ", $tal1 + $tal2;
?>
</body>
</html>
```

I exemplet ovan hämtar vi värdena till $tal1 och $tal2 från en speciell variabel som heter $_GET. I $_GET finns det data som ges till programmet via URL:en. Till exempel så kommer programmet om det anropas som http://dinserver/programnamnet.php?tal1=100&tal2=199 att skriva ut "Summan av talen 100 och 199 = 299".

Testa sedan andra tal och se vad som händer.

Du kanske provade att sätta ett av talen till någor annat än ett tal, till exempel bokstäver? I så fall såg du att PHP inte kan summera text. Observera att om du matar in ett decimaltal så måste decimalpunkt och inte komma användas.

6.2. Datatyper

Alla programmeringsspråk arbetar med olika datatyper. I PHP behöver du inte ange vilken typ det är du jobbar med utan det listar programtolken ut från innehållet. Det är dock väldigt viktigt att man håller reda på vilka typer ens variabler har eftersom det kan bli underliga fel annars.

Följande typer finns:

6.2.1. Skalära

- boolean, bool (true, false)
- integer, int (heltal)
- float, double, real (reella tal)
- string (Textsträngar)

6.2.2. Sammansatta

- arrayer (En samling av värden som indexeras antingen av en integer eller string).
- objekt (Kommer inte att ingå i denna kurs, se manualen om du är intresserad).

6.2.3. Övriga speciella

- NULL (Variabeln har inget värde alls).
- resource (Innehåller en referens till en extern tillgång).

6.3. Övningar

6.3.1. Addition

Skapa ett program som med hjälp av variabler summerar tre tal.

Kapitel 7. Operatorer

Detta avsnitt beskriver vad operatorer är och hur man använder dem i ett PHP-script.

7.1. Vad är en operator?

En operator är något som verkar på en eller flera termer. Exempel på operatorer är +, -, * och / som gör precis vad ni tror att de gör.

Det finns olika typer av operatorer. Aritmetiska operatorer, som de ovan, opererar bara på tal. Sedan finns det tilldelningsoperatorer som gör tilldelningar och så vidare. Här kommer några av de vi kommer att jobba med:

7.1.1. Aritmetiska operatorer

Tabell 7-1. Aritmetiska operatorer

Exempel	Namn	Resultat
$a + $b	Addition	Summan av $a och $b
$a - $b	Subtraktion	Differensen av $a och $b
$a * $b	Multiplikation	Produkten av $a och $b
$a / $b	Division	Kvoten av $a och $b
$a % $b	Modulus	Resten av division mellan $a och $b

7.1.2. Tilldelningsoperatorer

Det finns bara en tilldelningsoperator och den heter helt enkelt "tilldelas". Den representeras av ett lika-med-tecken (=). Så här kan den användas:

Exempel 7-1. Tilldelningsoperatorn

```
<?php
$c = $a + $b; // Utläses, c tilldelas värdet av a + b
?>
```

Det finns vissa andra tilldelningsoperatorer, men dessa behöver ni inte kunna. Jag tar dem kort här.

Exempel 7-2. Tilldelningsoperatorerna += och -=

```
<?php
$a += 5; // Samma sak som $a = $a + 5
$a -= 5; // Samma sak som $a = $a - 5
?>
```

7.1.3. Jämförelseoperatorer

Jämförelseoperatorer arbetar på tal och returnerar alltid ett värde av typen boolean. Det vill säga true eller false.

Tabell 7-2. Jämförelseoperatorer

Exempel	Namn	Resultat
$a == $b	Lika med	Sant om $a är lika med $b.
$a != $b	Inte lika med	Sant om $a inte är lika med $b.
$a < $b	Mindre än	Sant om $a är mindre än $b.
$a > $b	Större än	Sant om $a är större än $b.
$a <= $b	Mindre än eller lika med	Sant om $a är mindre än eller lika med $b.
$a >= $b	Större än eller lika med	Sant om $a är större än eller lika med $b.
$a === $b	Identiska	Sant om $a är lika med $b och båda är av samma typ.
$a !== $b	Inte identiska	Sant om $a inte är lika med $b eller om de inte är av samma typ.

7.1.4. Logiska operatorer

Som jag sade ovan så returnerar alltid de jämförande operatorerna av typen boolean och opererar på tal.

Logiska operatorer returnerar alltid boolean, men opererar också bara på logiska termer.

Tabell 7-3. Logiska operatorer

Exempel	Namn	Resultat
$a and $b	Och	Sant om $a och $b är sanna.
$a or $b	Eller	Sant om $a eller $b är sanna.
$a xor $b	Exklusivt eller	Sant om $a eller $b är sanna men inte båda två.
!$a	Inte/Icke	Sant om $a inte är sant.

7.1.5. Strängoperatorer

Det finns tre operatorer som opererar på strängar, den första känner ni till sedan förut och det är tilldelningsoperatorn "tilldelas". Den fungerar lika på strängar som på tal. Sedan finns det två till. Dessa beskrivs lättast med ett exempel:

Exempel 7-3. Strängoperatorer

```
<?php
$s = "Kalle ";    //$a tilldelas "Kalle "
$t = $a . "Anka"; //$t innehåller nu "Kalle Anka"
$s = "Kalle ";
$s .= "Anka";     //Samma sak som $s = $s . "Anka"
?>
```

7.2. Mer läsning

Det står mycket om operatorer i PHP-Manualen (*http://www.php.net/manual/en/language.operators.php* (http://www.php.net/manual/en/language.operators.php)).

Kapitel 8. Selektioner (Villkorssatser)

I de allra flesta programmeringsspråk finns det selektionssatster. Precis som namnet antyder handlar det om val. Programmet kan ta olika vägar beroende på olika villkor. Vanliga selektionssatser är if-satsen och if-else-satsen.

8.1. If-satsen

If-satser finns i de allra flesta språk och ser nästan likadan ut i dem alla. If-satsen fungerar som så att om något är sant så gör en sak, annars inte. Ett exempel är på sin plats.

Antag att jag vill att ett program skall tala om för mig om ett tal är större än 100. Jag vill att programmet skall skriva ut det tal jag anger och om det är större än 100 så skall det också skrivas ut. Så här kan det se ut:

Exempel 8-1. Större än 100

```
<html>
<head>
   <title>Större än 100</title>
</head>
<body>
   <?php
      echo "Du angav tal: $tal";

      if ($tal > 100) {
         echo "<p><emphasis>$tal är större än 100</emphasis></p>";
      }
   ?>
</body>
</html>
```

If-satsen består alltså av ordet if följt av ett test inom parenteser. Satsen som följer efter utförs om testet blir sant. Vill man att det skall vara flera satser som utförs om testet blir sant kan man slå ihop dem till ett block med hjälp av { och } (måsvingar). I exemplet ovan använder jag måsvingarna fast de egentligen inte behövs. Som regel är det bäst att alltid sätta dit måsvingarna ifall man vill stoppa in en rad till inom if-satsen sedan så glömmer man dem inte.

Studera nu if-satsen ovan och skriv om programmet och testa olika tal.

8.2. If-else-satsen

Nu fungerar programmet så långt. Men det vore ju kul om programmet sade till även om talet inte är större än 100. Alltså om det är större än 100 skriv det annars skriv att det inte är större än 100.

Studera följande exempel

Exempel 8-2. If-else exempel

```
<html>
<head>
   <title>Större än 100</title>
</head>
<body>
<?php
echo "Du angav tal: $tal";

if ($tal > 100) {
   echo "<p><emphasis>$tal är större än 100</emphasis></p>";
} else {
   echo "<p><emphasis>$tal är inte större än 100</emphasis></p>";
}
?>
</body>
</html>
```

I exemplet ovan ser vi hur en if-else sats fungerar. Om uttrycket inom parenteserna är sant utförs det som kommer efter. Om inte så utförs det som kommer efter else. *Det kan aldrig inträffa att båda satserna utförs!*

8.3. if-elseif

If-elseif är också en vanlig konstruktion. Den används ofta tillsammans med else och blir då en if-elseif-else sats. Man kan ha flera elseif i en konstruktion men bara en else. Det som kommer efter det första sanna uttrycket utförs och inget annat. Om inget är sant kommer det som står efter else (annars) att utföras.

Vi tittar på vårt exempel igen. Talet man anger kan ju vara större eller mindre än 100. Är det inte det så måste det ju vara talet 100 som angetts. Vi testar igen.

Exempel 8-3. if-elseif-else

```
<html>
<head>
```

```
   <title>Större än 100</title>
</head>
<body>
<?php
echo "Du angav tal: $tal";

if ($tal > 100) {
   echo "<p><emphasis>$tal är större än 100</emphasis></p>";
} elseif ($tal < 100) {
   echo "<p><emphasis>$tal är inte större än 100</emphasis></p>";
} else {
   echo "<p><emphasis>$tal är ju lika med 100</emphasis></p>";
}
?>

</body>
</html>
```

Som vi ser så är den inte helt olik de andra konstruktionerna med if. Den fungerar så att om det första testet är sant så utförs satsen efter det. Om inte görs testen efter elseif. Är denna sann så utförs satsen efter den. Om inget test har varit sant utförs det som kommer efter else. Det kan vara flera elseif och else kan utelämnas.

8.4. Mer läsning

Aktuellt avsnitt i PHP-manualen. *http://www.php.net/manual/en/control-structures.php* (http://www.php.net/manual/en/control-structures.php)

Kapitel 9. Iterationer (Upprepningar, loopar)

Datorprogram är extremt bra på att göra saker om och om igen, utan att ledsna eller göra fel. Till detta använder man någon typ av iterationssats (iteration = upprepning).

9.1. While-satsen

While-satsen är en vanlig iterationssats. Den fungerar så att en sats (som kan vara ett block) körs om och om igen så länge som ett test är sant. Studera följande exempel som skriver ut tiotusen ettor.

Exempel 9-1. Exempel med while

```
<?php
// Exempel på while-loop

echo "<h1>Tiotusen ettor</h1>";

$a = 0;
while ($a < 10000) {
   echo "1 ";
   $a = $a + 1;
}
?>
```

Vi tittar på exemplet rad för rad. Raden som börjar med "//" är en kommentar, den kan ni ignorera, kommentarer är viktiga men vi kommer att gå igenom dem lite senare.

Den andra raden är starten på while-loopen (kallas även loop eftersom den loopar om och om igen). Så länge som testet ($a < 10000) är sant so kommer satsen efter att repeteras. När $a är större än eller lika med 10000 kommer loopen att avbryts. Om man i loopen glömmer att öka $a kommer testet alltid att vara sant och man kommer aldrig ur loopen. Detta kallas för en oändlig loop och är ett vanligt programmeringsfel som gör att programmet hänger sig eller kraschar.

Om testet inte är sant från början så kommer aldrig det står i satsen att köras. Se därför till att testet är sant från början.

9.2. do-while-satsen

Do-while liknar på många sätt den vanliga while-satsen. Den enda skillnaden är att det som står i satsen alltid kommer att utföras åtminstone en gång. Se följande exempel:

Exempel 9-2. Exempel med do-while

```
<?php
// Ett exempel på hur man använder do-while

$i = 0;

do {
   echo "$i ";
   $i = $i + 1;
} while ($i < 100);

?>
```

Tilldelningen till $i är viktig eftersom den sätter värdet där loopen skall börja. I loopen skrivs först $i ut och sedan ökas variabeln $i med ett. Detta sker så länge som $i är mindre än 100. Alltså från 0 till 99.

9.3. for-loopen

For är den vanligaste iterationen. Den är dock vid en första anblick lite krångligare än de andra. Man kan om man vill använda while istället för for om man vill, men när man kan for är den mycket smidigare.

for-loopen skriver man med det reserverade ordet for följt av en parentes. Inom parentesen skall det stå tre stycken uttryck. Dessa tre skall se ut enligt följande.

- Det första kommer att exekveras en gång inan loopen börjar.
- Det andra skall vara ett boolskt uttryck. Loopen kommer att gå så länge detta är sant.
- Det tredje körs efter varje gång som loopen har gått.

Nu känner jag att exempel är på sin plats igen:

Exempel 9-3. Exempel med for-loop

```
<?php
//Exempel på for-loop
```

```
for ($i = 0; $i <= 10; $i++) {
   echo "$i<br>\n";
}
?>
```

Oftast används de tre olika satserna på precis det sätt som visas ovan. Nämligen att initiera en räknare, kolla ett gränsvärde och räkna upp räknaren, men inget hindrar att man använder dem på andra sätt.

9.4. Mer läsning

Aktuellt avsnitt i PHP-manualen: *http://www.php.net/manual/en/control-structures.php* (http://www.php.net/manual/en/control-structures.php)

Kapitel 10. Pseudokod

Kommer ...

10.1. Mer läsning

Länkar till mera läsning för den intresserade

Kapitel 11. Indentering

Att indentera sin kod är något man gör för att den skall bli lättare att läsa och lättare att hitta fel. Detta avsnitt beskriver hur man indenterar på ett bra sätt.

11.1. Vad är indentering?

Indentering går ut på att man med hjälp av olika mycket blanksteg (space) till vänster om koden kan på ett logiskt sätt gruppera koden så att den går lättare att läsa. Det finns flera olika sätt att intendera på och varje programmerare har sin egen stil. För att kod skall bli enhetliga så har många företag en kodstandard i vilken det beskrivs hur kommentering och indentering skall göra inom företaget. Det gör att alla programmerare känner sig hemma i varandras kod och att den totala kodmassan blir enhetlig och lättare att granska.

Grundprincipen är att kod som hänger ihop skall ha samma indenteringsnivå. Se följande exemepel:

Exempel 11-1. Indentering

```
<?php
if ($tal == 100) {
   echo "Talet är 100";
   $tal = $tal + 1;
}
?>
```

I exemplet ser vi att det som hör till if-satsen har flyttats in en nivå. Det gör det lätt att se att det hör till if-satsen och att måsvingarna är riktiga. Vissa indenterar if-satsen så här:

Exempel 11-2. Indentering

```
<?php
if ($tal == 100)
{
   echo "Talet är 100";
   $tal = $tal + 1;
}
?>
```

Om vi säger att vi har nästlade if-satser så syns det ännu tydligare vad bra det är att indentera.

Exempel 11-3. Indentering

```
<?php
if ($inloggad) {
   if ($tal == 100) {
      echo "Tal är hundra";
   } else {
      echo "Tal är inte hundra";
   }
} else {
   echo "Du är inte inloggad!"
}
?>
```

11.2. Mer läsning

Länkar till mera läsning för den intresserade

Svensk text med massor av exempel på indentering och kommentering, samt länkar till mer +info på engelska: http://www.phpsidan.nu/res_articles.php?view=art&id=48

Kapitel 12. Kommentering

Alla som någon gång jobbat i ett programmeringsprojekt vet att det är av yttersta vikt att man kommenterar sin kod. Detta avsnitt beskriver hur man kommenterar och vad man skall tänka på när man kommenterar sin kod.

12.1. Hur ser en kommentar ut

I PHP finns det två typer av kommentarer. De är *// Kommentar* och */* Kommentar **/*. Den första fungerar så att allt som kommer efter *//* och fram till radens slut är en kommentar och kommer att ignoreras av PHP-tolkaren. Den andra typen av kommentar fungerar så att det som står mellan */** och **/* är kommentarer. Den andra varianten kan sträcka sig över flera rader.

Exempel 12-1. Exempel med kommentering

```
<?php
/* Detta är en kommentar */
// Detta är en kommentar
$i = 1000; // Detta är också en kommetar
?>
```

12.2. Att kommentera sin kod

Att kommentera i sin kod är en konst. Det är mycket att tänka på. Det som är svårast är att veta hur mycket man skall kommentera. Det är lika illa att kommentera för mycket som för lite. Här kommer några riktlinjer.

Skriv i kommentaren VAD som görs och inte HUR det görs. Hur det görs skall koden i sig själv förklara.

Kommentera i en sammanhängande längre kommentar före ett avancerat block vad som görs. Ett litet exempel:

Exempel 12-2. Längre kommentar före block

```
<?php
//
```

```
// Nedanstående räknar ut summan av alla tal mellan tal1 och
// tal2.
//
// Summan skrivs ut och tal2 måste vara större än tal1
//

$summa = 0;
for ($i = $tal1; $i <= $tal2; $i++) {
   $summa = summa + $i;
}
echo $summa;
?>
```

Jämför detta med nedanstående kod som är full av "Papegojkommentarer" (En papegoja brukar bara lära sig att upprepa det den hör).

Exempel 12-3. Papegojkommentarer

```
<?php
$summa = 0; // Summan sätts till 0

for ($i = $tal1; $i <= $tal2; $i++) { // Räkna upp i från $tal1 till $tal2
   $summa = summa + $i; // Aktuellt tal läggs till summan
}
echo $summa; // Skriv ut summan
?>
```

Observera att det är svårare att förstå vad den här koden gör än den ovanför. Trots att den är full av kommantarer. Den nedre har bara kommentarer som beskriver vad koden i sig beskriver och tillför inget. Radkommentarer är tillåtet om de gör någon nytta. Till exempel då variabler deklareras är det bra att ha radkommentarer efter varje variabel där man beskriver vad man tänkt att variabeln skall göra.

12.3. Liten sammanfattning

- Kommentera inte för mycket och inte för litet.
- Koden i sig skall visa vad programmet gör.
- Beskriv gärna i en (längre) kommantar före en funktion eller avancerat block i en funktion vad det gör istället för att kommantera på varje rad.
- Kommentarer på samma rad som koden blir lätt "Papegojkommentarer" sådana är fula och skall inte göras.

Ett mer sammanhängande exempel finns nedan:

Exempel 12-4. Kommentering sammanhängade exempel

```
<?php
//
// kommentering.php
//
// Detta är ett litet skript som bara demonstrerar kommentering.
// I början av varje fil är det väldigt bra om man har ett block
// som detta där det står vad som finns i filen. Och hur man får
// tag i programmeraren.
//
// Av: Marcus Rejås <marcus@rejas.se>
// Ver: 1.002
//

//
// Följande visar hur användarens browser presenterar sig. Det
// är bra att före avancerade block eller funktioner i koden
// beskriva vad koden gör.
//

echo "Din browser presenterar sig som:<br>";
echo $HTTP_USER_AGENT;


//
// Man kan även visa vilket IP de kommer från
//
echo "<p>Du har IP-nummer:<br>";
echo $REMOTE_ADDR;

//
// Skriver ut alla tal mellan 1 och 10
//
echo "<p>Alla tal mellan 1 och 10 ";
$tal = 1;
while ($tal <= 10) {
   echo "$tal ";
   $tal++;
}

// Nedan visas samma kod med "Papegojkommentarer"
echo "<p>Alla tal mellan 1 och 10 ";
$tal = 1;               // Tal tilldelas 1
while ($tal <= 10) {    // Så länge som tal <= 10
   echo "$tal ";       // Skriv ut tal
   $tal++;             // Öka tal med ett
}

// Vilket går lättast att förstå?
```

```
// Exempel på en block-kommentar. Nedanstående är helt
// bortkommenterat

/*
  Nedan visas samma kod med "Papegojkommentarer"
$tal = 1;               // Tal tilldelas 1
while ($tal <= 10) {    // Så länge som tal <= 10
   echo "$tal ";       // Skriv ut tal
   $tal++;             // Öka tal med ett
}
*/

echo "<hr>Detta är bara ett skript som demonstrerar kommentering.
Titta på källkoden istället.";

?>
```

12.4. Mer läsning

Länkar till mera läsning för den intresserade

Svensk text med massor av exempel på indentering och kommentering, samt länkar till mer info på engelska: http://www.phpsidan.nu/res_articles.php?view=art&id=48

Kapitel 13. Mer om strängar

Strängar är en typ som består av följder av tecken. Till exempel så är "Jag heter Marcus" en sträng. I detta avsnitt tittar vi lite mer på vad man kan göra med strängar.

13.1. Vad är en sträng

En sträng är en grupp av tecken. Strängar förekommer, i stort sett, i alla program. PHP är ett språk som är väldigt rikt på funktioner för att hantera strängar. Mycket beroende på att det är ett språk för web-programmering där i princip allt som kommer från programmet är strängar. Det uppmuntrar även till att man låter okända användare mata in strängar till programmen vilket gör att man av säkerhetsskäl måste vara försiktig med strängarna.

I PHP markeras en sträng av att den innesluts av enkla eller dubbla citationstecken. Skillnaden är den att inom dubbla citationstecken kommer alla variabler i strängen att bytas ut mot sitt värde. Se följande exempel:

Exempel 13-1. Exempel med strängar

```
<?php
$summa = 1 + 6;
echo "Summan är $summa"; // Skriver ut: Summan är 7
echo ’Summan är $summa’; // Skriver ut: Summan är $summa
?>
```

Som du ser så sker ingen variabelsubstitution i den andra raden eftersom den omges av enkla citationstecken.

13.2. Escape-tecken

Som vi såg i förra stycket så omges en sträng av citationstecken. En naturlig fråga man då ställer sig är vad som händer om jag vill ha citationstecken i en sträng. Se följande exempel:

Exempel 13-2. Citationstecken i strängar

```
<?php
echo "Tjenare din gamle "hacker"";
?>
```

Man ser direkt att det inte kommer att bli bra. Hur skall tolkaren kunna veta var strängen slutar? Det som kommer att ske är att strängen börjar vid det första citationstecknet och slutar vid den andra. Den bosktav (h) som kommer efter kommer att orsaka ett "parse error". Hur gör man då? Jo om man vill infoga specialtecken i en sträng måste dessa föregås av specialtecknet \ (bakvänt snedstreck eller backslash). Strängen ovan blir då:

Exempel 13-3. Exempel på escape-tecken

```
<?php
echo "Tjenare din gamle \"hacker\"";
?>
```

Nu blir utskrifter som vi tänkt oss. Det finns även andra specialtecken:

Tabell 13-1. Specialtecken i strängar

Teckenkombination	Skrivs ut som
\"	"
\’	’
\\	\
\$	$
\n	Ny rad
\t	Tab

I strängar inom enkla citationstecken (’) så substitueras bara "\’". Alla andra representerar sig själva.

13.3. Lägga ihop strängar

Man kan inte lägga ihop strängar med hjälp av additionsoperatorn (+). Den är ju till för aritmetiska termer. Som tur är så finns det speciella opteratorer för just strängar. Den som lägger ihop två strängar kallas för concatenationsoperatorn. Den representeras av tecknet "." (punkt). Se nedanstående exempel.

Exempel 13-4. Exempel med strängar

```
<?php
$a = "Hello ";
$b = $a . "World!"; // Vi lägger till strängen "World!" efter $a
echo $b; // Skriver ut "Hello World!"
?>
```

13.4. Jämföra strängar

I PHP kan man jämföra strängar med de operaorer som vi lärt oss för numeriska värden. Det är ganska specifikt för PHP. Räkna inte med att du kan göra så i andra språk du kommer i kontakt med. Även i PHP finns det funktioner för att jämföra strängar.

13.4.1. Strcmp och strcasecmp

Strcmp (STRing CoMPare) är en funktion som jämför två strängar med varandra. Om de är exakt likadana returneras värdet 0. Om den första är större returneras 1 och om den andra är större returneras -1. Syntaxen och ett exempel på hur den kan användas visas i nedanstående exempel.

Exempel 13-5. Exempel med strcmp

```
<?php
if (strcmp($password, "Hemligt") == 0) {
   echo "Rätt lösenord";
} else {
   echo "Fel lösenord!";
}
?>
```

Tänk på att strcmp gör skillnad på stora och små bokstävar. Texten "Rätt lösenord" ovan kommer bara att skrivas ut om $password innehåller exakt "Hemligt". Vill du jämföra strängen utan att versaler/gemener skall ha någon betydelse kan du prova strcasecmp som fungerar på samma sätt fast "case insensitive".

13.5. Ändra strängar

Ofta vill man ändra på strängar så att de ser lite annorlunda ut. Det kan vara att man vill göra om alla bokstäver till versaler eller gemener. Eller att man vill byta något ord mot ett annat. PHP har massor av funktioner för detta. Vanliga saker man vill göra med strängar är att byta ut en förekomst av ett ord mot ett annat eller att göra alla tecken till små eller stora bokstäver eller ta bort tomma tecken.

13.5.1. strtoupper och strtolower

Dessa två funktioner (STRing TO UPPERcase och LOWERcase) tar en sträng som argument och returnerar samma sträng med alla bokstäver konverterade till antingen stora bokstäver (versaler) eller små bokstäver (gemener).

Se följande lilla exempel:

Exempel 13-6. Exempel med strtoupper och strtolower

```
<?php
$str1 = "Kalle Anka";
$str2 = strtolower($str1);  // str2 blir "kalle anka"
$str3 = strtoupper($str1);  // str3 blir "KALLE ANKA"
echo "\$str1: ". $str1;
echo "\n\$str2: ". $str2;
echo "\n\$str3: ". $str3 ."\n";
?>
```

13.5.2. Ucfirst och ucwords

Dessa två funtioner (UpperCase FIRST och WORDS) är kanske inte lika användbara som strtoupper och strtolower men kan vara bra att känna till. De kan användas till exempel om man vill snygga till användarinmatad data. Vad de gör är att göra den första bokstaven i en sträng (ucfirst) eller första bokstaven i varje ord i strängen (ucwords) till versal. Observera att dessa funktioner bara verkar på de tecken det gäller och inte de övriga. Se följande exempel:

Exempel 13-7. Exempel med ucfirst och ucword

```
<?php
$str1 = "kalle anka";
$str2 = ucwords($str1); // str2 blir "Kalle Anka"

$str1 = 'KALLE ANKA';
$str2 = ucwords($str1); // str2 blir KALLE ANKA (inga tecken blir gemener)

$str3 = ucwords(strtolower($str1)); // str3 blir Kalle Anka
?>
```

I exemplet ser vi att "KALLE ANKA" blir samma sak efter ucwords. Det beror på att denna funktion bara verkar på det första tecknet i varje ord. Detta görs versalt. Inget annat görs. Är alla tecken versaler så kommer inget att ske. I den sista satsen lägger vi in ett anrop till strtolower vilket gör att ucwords matas med strängen "kalle anka" istället.

Ucfirst fungerar på exakt samma sätt fast bara på det allra första tecknet i strängen.

13.5.3. strrev

Detta är en väldigt trevlig lite funktion (STRing REVerse). Det enda den gör är att den tar en sträng som argument och returnerar samma sträng fast reverserad.

Exempel 13-8. Exempel med strrev

```
<?php
$str1 = "Kalle Anka";
$str2 = strrev($str1); // str2 blir "aknA ellaK"
?>
```

13.5.4. strlen

En betydligt mer användbar funktion än strrev är strlen (STRing LENgth). Denna returnerar antalet tecken i en sträng.

Som vanligt tar vi ett litet exempel:

Exempel 13-9. Exemepel med strrev

```
<?php
$str1 = "Kalle Anka";
echo strlen($str1);          // skriver ut 10
echo strlen(" Kalle Anka "); // skriver ut 12
?>
```

13.5.5. str_replace

Detta är en mycket användbar funktion. Den byter ut en teckenföljd i en mening mot en annan. Funktionen tar tre argument och returnerar en sträng. Det första argumentet är den teckenföljd som skall ersättas, det andra är det som det skall ersättas med och det tredje argumentet är den sträng som det hela berör. Det som returneras är strängen i det tredje argumentet där alla förekomster av teckenföljden i det första argumentet ersatts med tecknen i det andra argumentet.

Förvirrad? Se nedanstående exempel:

Exempel 13-10. Exempel med str_replace

```
<?php
$str1 = "Kalle Anka är bäst";
$str2 = str_replace("Kalle", "Kajsa", $str1);

echo $str2;  // str2 är "Kajsa Anka är bäst"
?>
```

Detta är användbart till väldigt mycket. Bara fantasin sätter gränserna.

13.6. Hantera oönskad HTML i strängar

I PHP hanterar man av naturliga skäl ofta strängar som skall presenteras i en webläsare. Flera av dessa strängar kommer vanligtvis från användare eller andra osäkra källor. Då vill man gärna kontrollera så att inte användaren kan mata in data som förstör resten av sidan. Till exempel så skall man inte i ett web-forum kunna skriva in HTML-kod hur som helst. Man skulle då kunna länka in fula bilder eller andra typsnittet på hela sidan. Man kan åstadkomma detta genom att använda massor av anrop på str_replace. Till exempel:

Exempel 13-11. Med str_replace

```
<?php
//
// Följande två rader tar bort alla förekomster av större än och
// mindre än och ersätter dem med deras HTML-motsvarighet. Detta
// eliminerar alla HTML-taggar.
//
$html_string = str_replace("&lt;", "&amp;lt;", $html_string);
$html_string = str_replace("&gt;", "&amp;gt;", $html_string);
?>
```

Men det finns bättre sätt att göra det på. Det finns (naturligtvis) färdiga funktioner som gör HTML av strängar, till exempel htmlentities. htmlentities gör om en sträng så att alla tecken som har en motsvarighet i HTML kod blir just, HTML kod. Funktionen tar ett argument och har två stycken valfria argument som du inte behöver ange, mer än om du t.ex. behöver använda en annan teckenuppsättning. Se följande exempel:

Exempel 13-12. Exempel med htmlentities

```
<?php
$str1 = "<h1>Kalle Anka</h1>"; // Blir "Kalle Anka" (Rubrik 1) i browsern
$str2 = htmlentities($str1);   // Blir <h1>Kalle Anka</h1>; i browsern
```

```
?>
```

Detta gör att allt man skickar till htmlentities kommer att synas i browsern precis som det ser ut i klartext.

13.7. Söka i strängar

Ofta vill man hitta eller använda bara vissa delar av en sträng. Det finns flera funktioner för detta. Vi skall titta på två av dem.

13.7.1. strstr och stristr

Dessa två funktioner fungerar så att de tar två argument, båda är strängar (STRing in STRing och STRing case-Insensitive in STRing). Den första är den sträng som det skall sökas i och det andra är det sökta. Det som returneras är det som är kvar av det första argumentet efter den funna strängen. Förvirrad?

Exempel 13-13. Exemempel med strstr()

```
<?php
$namn = "Kalle Anka";
$enamn = strstr($namn, " ");    // enamn blir " Anka"

$adress = "kalle@ankeborg.net";
$domain = strstr($adress, "@"); // domain blir "@ankeborg.net"
?>
```

Funktionen stristr fungerar på samma sätt men den bryr sig inte om om den eftersökta strängen har stora eller små bokstäver (Case-Insensitive).

13.8. Mer läsning

Aktuellt avsnitt i Manualen. *http://www.php.net/manual/en/ref.strings.php* (http://www.php.net/manual/en/ref.strings.php)

Kapitel 14. Funktioner

Funktioner används för att dela upp programmet i mindre små delar. Det gör att programmeraren kan koncentrera sig på en del i taget och löper mindre risk att göra fel. Man använder även funktioner till kod som kan återfinnas på flera ställen för att minska duplikation av kod.

14.1. Vad är en funktion?

En funktion är ett antal instruktioner som fristående utför en sak. Denna snutt kan sedan köras om och om i samma program eller delas med andra program. Ett exempel:

Exempel 14-1. Funktioner

```
<?php
/*
 * print_html_header_start
 *
 * Skriver ut en html-header
 */
function print_html_header_start () {
   echo "<html><head></head><body>";
}

/*
 * print_html_header_stop
 *
 * Skriver ut slut-html-taggar
 */
function print_html_header_stop () {
   echo "</body></html>";
}


// Här börjar programmet
print_html_header_start();

// Andra utskrifter här.
print_html_header_stop();
?>
```

I exemplet ovan deklareras två funktioner som skriver ut en html-header och en html-footer. Dessa funktioner anropas längre ned i programmet.

14.2. När skall man använda funktioner?

Man skall använda funktioner så snart man kan. Själva programmet brukar ofta vara bara anrop till olika funktioner. Följande kan vara bra grundregler.

Använd funktioner till:

- Alla uppgifter som kan avgränsas
- Alla uppgifter som är repititiva
- All kod som du kan tänkas återanvända i andra program

Försök att tänka ett program i funktioner.

Vi funderar på programmet password.php och försöker identifiera olika funktioner.

En funktion är att skriva ut html-formuläret. Det är ju en avgränsad uppgift. En annan solklar funktion är autentiseringen.

14.3. Argument till funktioner

En funktion kan ta noll eller flera argument. Ett argument är ett värde som man skickar till funktionen. Till exempel om man vill vill göra en funktion som summerar två tal så är det bra om den kan ta de två talen som argument. Se följande exempel:

Exempel 14-2. Argument till funktioner

```
<?php
function summa($tal1, $tal2) {
        return ($tal1 + $tal2);
}

echo summa(5,6);
?>
```

Observera att ordningen på argumenten spelar roll. I exemplet kommer tal1 att bli 5 och tal2 6. Variablerna $tal1 och $tal2 existerar bara inom funktionen och inte i resten av programmet. Vilken som får vilket värde bestäms av ordningen i funktionsanropet.

14.4. Returvärden

I funderingen ovan vore det ju bra om autentiseringen kunde returnera ett värde (ett boolskt värde till exempel). Det kan se ut så här:

Exempel 14-3. Retur från funktioner

```
<?php
function is_logged_in ($name, $pass) {
        if ($name == "kalle") && ($pass == "ankeborg")) {
                return true;
        } else {
                return false;
        }
}
?>
```

Denna funktion är ganska kompakt och gör sig fötjänt av en förklaring. Den tar två argument, $namn och $pass. Funktionen returnerar värdet av en boolsk operation.

14.5. Mer läsning

Aktuellt avsnitt i manualen. *http://www.php.net/manual/en/functions.php* (http://www.php.net/manual/en/functions.php)

Kapitel 15. Filer och filhantering

När man skriver datorprogram vill man ofta spara information mellan olika körningar program. Ett sätt att göra detta är att lagra informationen i en eller flera filer. I detta kapitel behandlas hur man gör.

15.1. Filer

Innan vi börjar använda oss av filer i programmeringen skall vi titta lite kort på vad en fil är. En fil är en samling sammanhängande information på ett medium, oftast en hårddisk, som man namngett.

För att läsa eller skriva i filen använder man ett filhandtag (eng. file handle). När man programmerar kan man inte hoppa runt i filen hur som helst lika enkelt som man gör i till exempel ett ordbehandlingsprogram.

15.2. Arbetsmetod vid arbete med filer

När man jobbar med filer i PHP använder man följande metodik.

Man öppnar en fil med ett anrop till funktionen *fopen* . Den funktionen returnerar ett filhandtag. Detta filhandtag kan man sedan använda för att skriva till eller läsa från filen. Slutligen skall man stänga sin fil med funktionen *fclose*. När man har en fil öppen finns det också, även om man inte märker det så ofta, en så kallad filpekare som håller reda på var i filen man är.

15.3. Funktionen fopen

Funktionen fopen tar två argument och returnerar ett filhandtag. Så här kan ett anrop till fopen se ut:

Exempel 15-1. Funktionen fopen

```
<?php
$filhandtag = fopen("/home/rejas/data/testfil", "a");
?>
```

Vad som sker är att $filhandtag tilldelas ett handtag till filen testfil i katalogen /home/rejas/data/. Observera att detta är sökvägen till filen på servern och att den användare som kör webservern måsta ha rätt att läsa och eventuellt skriva till filen. Det första argumentet är således filnamnet, men vad är det

andra? Jo det andra talar om på vilket sätt filen skall öppnas. Följande är de vanligaste värdena på det andra argumentet och deras innebörder:

Tabell 15-1. Andra argumentet till fopen

Värde	Innebörd
r	Öppnar en fil endast för läsning, filpekaren placeras först i filen.
r+	Öppnar en fil för läsning och skrivning, filpekaren placeras i början av filen.
w	Öppnar en fil endast för skrivning, filpekaren ställs ställs först i filen. Om filen inte finns skapas den och om den finns så blir den överskriven.
w+	Samma som w men öppnar även för läsning
a	Öppnar endast för skrivning. Skapar filen om den inte finns. Ställer filpekaren i slutet av filen.
a+	Samma som a men även för läsning.
x	Öppnar en fil för skrivning och placerar filpekaren i början av filen. Om filen redan existerar returnerar funktionen FALSE och ett varnings meddelande kan komma att skrivas ut. Annars skapas filen.
x+	Samma som x men tillåter även läsning av filen.

Argumenten med + till fopen verkar bra att använda men används faktiskt inte så ofta som man kan tro.

15.4. Funktionen fwrite

Funktionen fwrite använda för att skriva till en fil. Den tar två argument, ett filhandtag och så det som skall skrivas till filen.

Ett exmpel:

Exempel 15-2. Funktionen fwrite

```
<?php
$fh = fopen("/home/rejas/data/testfil", "a");
fwrite($fh, "Hej på dig\n");
fclose($fh);
?>
```

Funktionen fwrite returnerar false om det skulle vara så att den inte kan skriva till filen så det kan vara bra att kolla att det går bra.

Exempel 15-3. Funktionen fwrite med felkontrol

```
<?php
$fh = fopen("/home/rejas/data/testfil", "a");
if (! fwrite($fh, "Hej på dig\n")) {
   echo "Ooops, fel vid skrivning till fil";
   exit; // Avbryter körningen
}

fclose($fh);
?>
```

Även fopen returnerar false om den inte kan öppna filen, att kolla detta lämnas som övning till läsaren :).

15.5. Readfile

Readfile är en av många funktioner som kan användas för att läsa från en fil. Den läser en hel fil och skriver ut den på utskiftsbufferten.

Se följande exempel:

Exempel 15-4. Exempel med readfile

```
<?php
readfile("/home/rejas/data/testfil");
?>
```

15.6. Sammanhängande exempel

Nu kan vi skriva ett litet program som varje gång det körs lägger till en rad i en fil och skriver ut filen.

Exempel 15-5. Sammanhängande exempel på filanvändning

```
<?php
$filename = "/home/rejas/data/testfil";

$fh = fopen($filename, "a");

if (! fwrite($fh, "Hej på dig\n")) {
   echo "Ooops, fel vid skrivning till fil";
```

```
   exit; // Avbryter körningen
}

fclose($fh);

readfile($filename);
?>
```

15.7. Mer läsning

Mer information om funktioner för att hantera filer och filsystem finns här:
http://www.php.net/manual/en/ref.filesystem.php (http://se.php.net/manual/en/ref.filesystem.php)

Kapitel 16. Arrayer (vektorer)

Vi har tidigare tittat på variabler. Nu skall vi titta på en speciell typ av variabler nämligen arrayer. En array är en variabel som kan innehålla flera olika värden. En array kallas iband även för en vektor.

16.1. Arrayer

Ibland vill man spara flera värden i en variabel. Till exempel om de måste höra ihop och inte får komma isär eller om man vill returnera flera världen från en funktion. Det är precis var arrayer är, en typ av variabel som kan hålla flera värden.

Vi kastar oss direkt in på ett exempel:

Exempel 16-1. Arrayer

```
<?php
// Vi skapar en array som heter arr och innehåller tre värden.
$arr[0] = 10;
$arr[1] = 20;
$arr[2] = 30;

echo $arr[2]; // Skriver ut 30
?>
```

16.2. Array-funktioner

En annan fördel med arrayer är att det finns massor av funktioner som verkar på arrayer. Till exempel så finns det funktioner för att sortera, reversera eller blanda arrayer.

Vi skall titta på ett par användbara funktioner:

16.2.1. Funktionen array_reverse

Den här funktionen tar en array som argument och returnerar samma array fast i omvänd ordning. Det vill säga det som var sist i arrayen tidigare liger först i den array som array_reverse returnerar. Ett litet exempel:

Exempel 16-2. Exempel med array_reverse()

```
<?php
$arr[0] = 1;
$arr[1] = 2;
$arr[2] = 3;

$arr2 = array_reverse($arr);

echo $arr2[0] ."\n"; // Skriver ut 3
echo $arr2[1] ."\n"; // Skriver ut 2
echo $arr2[2] ."\n"; // Skriver ut 1
?>
```

16.2.2. Funktionen sort

Funktionen sort tar en array som argument och sorterar den. Observera att denna funktion inte returnerar något utan sorterar den array den får som argument.

Exempel 16-3. Exempel med sort()

```
$arr2[0] = "Kalle";
$arr2[1] = "Fnatte";
$arr2[2] = "Knatte";
$arr2[3] = "Kajsa";
$arr2[4] = "Joakim";
$arr2[5] = "Alexander";
$arr2[6] = "Tjatte";


sort($arr2); // Observera att inget returneras, utan att arrayen blir sorterad.

for ($i = 0; $i <= 6; $i++) {
        echo "$arr2[$i]<br>";
}
```

16.2.3. Funktionen file

File är en funktion som kanske bättre pasar bland filfunktionerna (där finns den till exempel i PHP-manualen) men jag har valt att lägga den här eftersom den inbjuder till att använda andra array-funktioner.

File tar ett argument som skall vara ett filnamn, den returnerar en array med varje rad i filen i ett element.

16.3. Mer läsning

http://www.php.net/manual/en/language.types.array.php http://www.php.net/manual/en/ref.array.php

Kapitel 17. Mer om funktioner

Tidigare har vi lärt oss hur funktioner fungerar. I detta kapitel lär vi oss lite mer om dem.

17.1. Call by reference, call by value

När man anropar en funktion skickar man vanligtvis med ett eller flera argument. Vi har tidigare sett att om man ändrar dessa argument i funktionen så ändras de inte utanför funktionen. Det är för att man lättare skall kunna flytta funktionen mellan olika program utan att behöva vara rädd för att den skall ändra något utanför själva funktionen. Detta kallas för *call by value*. Det vill säga att värdet i variabeln skickas till funktionen, inte själva variabeln.

Ibland kan man vilja låta funktionen ändra variabler även i världen utanför. Alltså om jag skickar en variabel till en funktion och den ändras i funktionen så skall den ändras även utanför funktionen. Detta förfarande kallas för *call by reference*. Det vill säga att man istället för att skicka en kopia på värdet i en variabel så skickar man en pekare till variablen så att det man gör men variabeln i funktionen även händer utanför funktionen.

Detta kan i flera fall vara väldigt smidigt men skall normalt undvikas. Ett litet exempel.

Exempel 17-1. Call by reference

```
<?php
function swap (&$var1, &$var2) {
   $tmp = $var2;
   $var2 = $var1;
   $var1 = $tmp;
}

$text1 = "Text1";
$text2 = "Text2";

echo "Före swap: text1: $text1, text2: $text2 <br>\n";

swap($text1, $text2);

echo "Efter swap: text1: $text1, text2: $text2 <br>\n";
?>
```

Kapitel 18. Inmatning

Kommer ...

Appendix A. Kurs DTR1207 - Programmering A

50 poäng, inrättad 2000-07 SKOLFS: 2000:28

A.1. Mål

A.1.1. Mål för kursen

Kursen skall ge grundläggande teoretiska och praktiska kunskaper i programmering. Kursen skall även ge kunskaper om vanliga användningsområden för olika programmeringsspråk. Kursen skall också ge grundläggande färdigheter i systemering och struktureringsteknik.

A.1.2. Mål som eleverna skall ha uppnått efter avslutad kurs

Eleven skall

kunna något programmeringsspråks grundläggande datatyper, fördefinierade strukturer och funktioner samt deras regler och syntax

kunna analysera programmeringsuppgifter och formulera strukturerad pseudokod samt konstruera enkla algoritmer

kunna systemera och strukturera programmeringsarbetet samt skriva enkla program och felsöka källkod

känna till kompilatorns/länkarens arbete från källkod till färdigt program

känna till viktiga operativsystemstandarder för bl.a. teckenkoder och utmatningsrutiner

känna till språkens allmänna prestanda och egenskaper samt vilka programmeringsuppgifter de är bäst lämpade för.

A.2. Betygskriterier

A.2.1. Kriterier för betyget Godkänd

Eleven analyserar enkla programmeringsuppgifter och skapar med viss handledning körbara väldokumenterade program.

Eleven söker med viss handledning upp de fakta som behövs för programmeringsuppgifterna.

Eleven beskriver det använda programspråkets uppbyggnad, viktigaste funktioner, egenskaper och prestanda.

A.2.2. Kriterier för betyget Väl godkänd

Eleven utför sina programmeringsuppgifter på egen hand och inom rimlig tid.

Eleven hämtar på egen hand fakta från olika källor och tillämpar dessa i uppgifterna.

A.2.3. Kriterier för betyget Mycket väl godkänd

Eleven utför självständigt sina programmeringsuppgifter med noggrannhet och når snabbt avsett resultat.

Eleven anpassar sin arbetsinsats till situationen, analyserar resultat samt åtgärdar kvalitetsavvikelser.

Eleven beskriver samband och ser helheter i komplicerade programmeringssituationer.

Skolverket 2002-04-09

Appendix B. Kurs DTR1208 - Programmering B

50 poäng, inrättad 2000-07 SKOLFS: 2000:28

B.1. Mål

B.1.1. Mål för kursen

Kursen skall ge fördjupade teoretiska och praktiska kunskaper i ett strukturerat programmeringsspråk. Kursen skall också ge kunskaper om språkets viktigaste datastrukturer. Dessutom skall kursen ge färdigheter i algoritmkonstruktion.

B.1.2. Mål som eleverna skall ha uppnått efter avslutad kurs

Eleven skall

förstå och kunna använda språkets vanliga datastrukturer såsom fält, lista, stack och filsystem

kunna införa och använda array, länkade listor och trädstrukturer i datastrukturer

förstå och implementera vanliga sorteringsalgoritmer

förstå och implementera vanliga sökalgoritmer

kunna analysera programmeringsuppgifter och formulera strukturerad pseudokod

kunna skriva program och felsöka källkod

känna till kompilatorns/länkarens uppgift vid arbete från källkod till färdigt program

känna till viktiga operativsystemstandarder för bl.a. teckenkoder och utmatningsrutiner

känna till språkets allmänna prestanda och egenskaper samt vilka programmeringsuppgifter det är lämpligast för.

B.2. Betygskriterier

B.2.1. Kriterier för betyget Godkänd

Eleven analyserar programmeringsuppgifter och skapar med viss handledning enkla körbara väldokumenterade program.

Eleven söker med viss handledning upp de fakta som behövs för programmeringsuppgifterna.

Eleven beskriver det använda programspråkets uppbyggnad, viktigaste funktioner, egenskaper och prestanda.

B.2.2. Kriterier för betyget Väl godkänd

Eleven utför sina programmeringsuppgifter på egen hand och inom rimlig tid.

Eleven hämtar på egen hand fakta från olika källor och tillämpar dessa i uppgifterna.

B.2.3. Kriterier för betyget Mycket väl godkänd

Eleven utför självständigt sina programmeringsuppgifter med noggrannhet och når snabbt avsett resultat.

Eleven anpassar sin arbetsinsats till situationen, analyserar resultat samt åtgärdar kvalitetsavvikelser.

Eleven beskriver samband och ser helheter i komplicerade programmeringssituationer

B.3. Specialisering mot enligt lista nedan. För betyg mm. använd angivna koder.

Kod	Namn
PCOB1408	Cobol
PDEL1408	Delphi
PJAV1408	Java
PPAS1408	Pascal

Kod	**Namn**
PPER1408	Perl
PPHP1408	PHP
PROC1408	C++
PVBA1408	Visual Basic

Skolverket 2002-04-09

Appendix C. Reserverade ord i PHP

Reserverade ord har en speciell innebörd i PHP. Du kan inte använda dem som namn på konstanter, namn på klasser eller namn på funktioner. Du kan naturligtvis använda dem i strängar.

De reserverade orden är: *and, array() , as , break , case, cfunction , __CLASS__, class, const, continue, declare, default, die(), do, echo(), else, elseif, empty(), enddeclare, endfor, endforeach, endif, endswitch, endwhile, eval, exception, exit(), extends, __FILE__ , for, foreach, function, __FUNCTION__, global, if, include(), include_once(), isset(), __LINE__ , list(), __METHOD__, new, old_function, or , php_user_filter, print(), require(), require_once(), return(), static, switch, unset(), use, var, while, xor*

Appendix D. GNU Free Documentation License

Version 1.2, November 2002. Svensk översättning av Marcus Rejås och Alexander Nordström, Januari 2004.

This is an unofficial translation of the GNU Free Documentation License into Swedish. It was not published by the Free Software Foundation, and does not legally state the distribution terms for documentation that uses the GNU FDL -- only the original English text of the GNU FDL does that. However, we hope that this translation will help Swedish speakers understand the GNU FDL better.

Detta är en inofficiell översättning av GNU Free Documentation License till svenska. Den har inte publicerats av Free Software Foundation och är inte juridiskt gällande för spridning av dokumentation som använder GNU FDL -- bara den engelska originaltexten i GNU FDL gäller. Vi hoppas att denna översättning skall hjälpa svensktalande att förstå GNU FDL bättre.

Originalet Copyright (C) 2000,2001,2002 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA. Denna översättning Copyright (C) 2004, Svenska Linuxföreningen, info@se.linux.org. Var och en äger kopiera och sprida ordagranna kopior av detta licensavtal [originalet och denna översättning], men att ändra det är inte tillåtet.

D.1. 0. BAKGRUND

Syftet med denna licens är att göra en handbok, bok, eller annat praktiskt och användbart dokument "fritt" som i frihet: att försäkra var och en den faktiska friheten att kopiera och sprida det vidare, med eller utan förändringar, antingen kommersiellt eller ideellt. Sekundärt bevarar denna licens ett sätt för författaren och förläggaren att få ära för deras arbete utan att de anses vara ansvariga för förändringar gjorda av andra.

Denna Licens är en sorts "copyleft", vilket betyder att derivativa verk av detta dokument själva måste vara fria på samma sätt. Den kompletterar GNU General Public License, som är en copyleft-licens utformad för fri programvara.

Vi har utformat denna licens för att den skall användas för handböcker till fri programvara, eftersom fri programvara behöver fri dokumentation: ett fritt program bör ha en handbok som erbjuder samma friheter som programmet gör. Men denna licens är inte begränsad till programvaruhandböcker; den kan användas för vilket textverk som helst oavsett ämne eller huruvida det är en utgiven, tryckt bok. Vi rekommenderar denna licens huvudsakligen för alla verk vars syfte är instruktion eller referens.

D.2. 1. TILLÄMPNINGSOMRÅDE OCH DEFINITIONER

Denna licens [det engelska originalet] gäller för varje handbok eller annat verk, oavsett uttrycksform, som innehåller ett meddelande där upphovsrättsinnehavaren stadgat att verket kan spridas enligt villkoren i GNU Free Documentation License. Ett sådant meddelande ger en internationell frihet utan

krav på ersättning och utan tidsbegränsning att använda verket under villkoren i denna licens [det engelska originalet]. "Dokument" nedan syftar på godtycklig handbok eller verk. Var och en är licenstagare och benämns som "du". Du accepterar villkoren i GNU Free Documentation License om du kopierar, modifierar eller sprider verket på ett sådant sätt att det kräver tillstånd enligt gällande upphovsrättslagstiftning.

En "förändrad version" av dokumentet avser varje verk som innehåller dokumentet eller en del av det, antingen ordagranna kopior, eller med ändringar och/eller översatt till ett annat språk.

Ett "sekundärt avsnitt" är en märkt bilaga eller förord till dokumentet som exklusivt behandlar förhållandet mellan dokumentets förläggare eller författare och dokumentets huvudsakliga ämne (eller till relaterade ämnen) och som inte innehåller något som direkt faller under det huvudsakliga ämnet. (Således, om dokumentet delvis är en lärobok i matematik så får ett sekundärt avsnitt inte förklara någon matematik.) Förhållandet kan vara en historisk koppling till ämnet eller något relaterat, eller en juridisk, kommersiell, filosofisk, etisk eller politisk ställning till det.

De "oföränderliga avsnitten" är sekundära avsnitt vars titlar är angivna som oföränderliga avsnitt i meddelandet som stadgar att dokumentet är utgivet under denna licens [det engelska originalet]. Om ett avsnitt inte innefattas av den ovanstående definitionen av sekundärt så är det inte tillåtet att ange det som oföränderligt. Dokumentet behöver inte innehålla några oföränderliga avsnitt. Om dokumentet inte anger några oföränderliga avsnitt så finns det inga.

"Omslagstexterna" är speciella korta ordföljder som är listade som framsidestexter eller baksidestexter i meddelandet som stadgar att dokumentet är utgivet under denna licens [det engelska originalet]. En framsidestext kan vara som mest 5 ord och en baksidestext kan vara som mest 25 ord.

En "transparent" kopia av dokumentet är en maskinläsbar kopia, representerad i ett format vars specifikation finns tillgänglig för allmänheten, som lämpar sig för att revidera dokumentet på ett enkelt sätt med generella textredigeringsprogram eller (för pixelbaserade bilder) generella grafikprogram eller (för ritningar) något väl tillgängligt ritprogram, och som är passande som indata till textfomaterare eller för automatisk konvertering till en mängd format som passar som indata till textformaterare. En kopia i ett för övrigt transparent filformat vars markeringar, eller avsaknad av markeringar, har ordnats för att hindra eller motverka att vidare förändring vidtas av läsare är inte transparent. Ett bildformat är inte transparent om det används för någon betydande del text. En kopia som inte är "transparent" kallas "opak".

Exempel på passande format för transparenta kopior innefattar ren ASCII utan markeringar, Texinfo indataformat, LaTeX indataformat, SGML eller XML som använder en publikt tillgänglig DTD, och standardenlig HTML, PostScript eller PDF utformat för mänsklig förändring. Exempel på transparenta bildformat innefattar PNG, XCF och JPG. Opaka format innefattar leverantörsspecifika format som bara kan läsas och editeras med leverantörsspecifika ordbehandlare, SGML eller XML för vilket DTD och/eller verktyg för behandling inte finns allmänt tillgängliga, och den maskingenererade HTML, PostScript eller PDF som produceras av vissa ordbehandlare enbart avsett som utdata.

"Titelsidan" innebär, för en tryckt bok, titelsidan själv, och sådana därpå följande sidor som krävs för att göra det material som enligt denna licens skall synas på titelsidan läsbart. För verk i sådana format som inte har någon egentlig titelsida, avses med "titelsida" den text som är närmast den mest framstående förekomsten av verkets titel, föregående den huvudsakliga textmassan.

Ett avsnitt "med titeln ÅÄÖ (XYZ)" avser en namngiven del av dokumentet vars titel är exakt XYZ eller innehåller XYZ inom parentes efterföljande text som översätter XYZ till ett annat språk. (Här står XYZ för ett speciellt namn på ett avsnitt nedan, som till exempel "Acknowledgements", "Dedications", "Endorsements" eller "History" [och ÅÄÖ för lämplig översättning, till exempel "tillkännagivanden", "dedikationer", "endossering" respektive "historik"].) Att "bevara titeln" på ett sådant avsnitt när du ändrar dokumentet innebär att det förblir ett avsnitt "med titeln ÅÄÖ (XYZ)" enligt denna definition.

Dokumentet får innehålla garantiavsägelser invid meddelandet om att denna licens [det engelska originalet] gäller för dokumentet. Dessa garantiavsägelser skall anses vara inkluderade per referens i denna licens, men bara för att friskriva från garantier. All annan innebörd dessa garantiavsägelser kan ha är ogiltiga och påverkar inte på något sätt innebörden i denna licens.

D.3. 2. ORDAGRANN KOPIERING

Du äger kopiera och sprida dokumentet på valfritt medium, antingen kommersiellt eller ideellt, förutsatt att denna licens [det engelska originalet], upphovsrättsklausul, och meddelandet som stadgar att GNU Free Documentation License gäller för dokumentet finns med på alla kopior, och att du inte lägger till några som helst andra villkor än de som ingår i denna licens. Du äger inte vidta tekniska åtgärder för att begränsa eller kontrollera läsande eller vidare kopiering av de kopior du skapar eller sprider. Dock äger du ta emot kompensation i utbyte mot kopior. Om du sprider tillräckligt många kopior måste du också följa villkoren i paragraf 3.

Du äger också låna ut kopior, under samma villkor som ovan, och du äger visa kopior offentligt.

D.4. 3. OMFATTANDE KOPIERING

Om du publicerar tryckta kopior (eller kopior i medier som normalt har tryckta omslag) av dokumentet, i en upplaga överstigande 100 exemplar, och dokumentets licensmeddelande kräver omslagstexter, så måste du förse kopiorna med omslag som, klart och tydligt, visar alla omslagstexter: framsidestexter på framsidan och baksidestexter på baksidan. Båda omslagen måste klart och tydligt identifiera dig som utgivare av dessa kopior. Framsidan måste presentera dokumentets hela titel, med alla ord i titeln lika framträdande och synliga. Du äger lägga till ytterligare stoff på omslagen. Kopiering med förändringar gjorda bara på omslaget, så länge som de bevarar dokumentets titel och i övrigt uppfyller dessa krav kan anses vara ordagrann kopiering i andra avseenden.

Om de obligatoriska texterna för något omslag är för omfattande för att rymmas i läsbart skick skall du

placera de första (så många som får plats) på det egentliga omslaget, och fortsätta med resten på de direkt intilliggande sidorna.

Om du publicerar opaka kopior av dokumentet i upplagor om mer än 100, måste du antingen bifoga en maskinläsbar transparent kopia med varje opak kopia, eller ange i eller med varje opak kopia en nätverksadress som är tillgänglig för den allmänna nätverksanvändande massan där man, med öppet standardiserade protokoll, kan ladda ner en komplett transparent kopia av dokumentet, utan extra material. Om du väljer det senare alternativet, måste du vidta skäliga åtgärder, när du börjar sprida opaka kopior i kvantitet, för att denna transparenta kopia skall förbli tillgänglig på angivna platsen till åtminstone ett år efter den sista gången du spred en opak kopia (direkt eller via ombud eller återförsäljare) av den utgåvan till allmänheten.

Det är önskvärt, men inte ett krav, att du kontaktar författarna till dokumentet i god tid innan du sprider något större antal kopior, för att ge dem en chans att förse dig med en uppdaterad version av dokumentet.

D.5. 4. FÖRÄNDRINGAR

Du äger kopiera och sprida en förändrad version av dokumentet under de villkor som beskrivs i paragraf 2 och 3 av GNU Free Documentation License, förutsatt att du släpper den förändrade versionen under exakt denna licens, och att den förändrade versionen antar dokumentets roll, och således medger spridning och förändring av den förändrade versionen till envar som erhåller en kopia av den. Utöver detta måste du göra följande med den ändrade versionen:

A. På titelsidan (och omslagen om det finns några) använda en titel skild från den som [original]dokumentet har, och skild från tidigare versioners titel (som skall, om det finns några, finnas listade i historikavsnittet i dokumentet). Du äger använda samma titel som det föregående dokumentet om den ursprungliga utgivaren ger sitt tillstånd.

B. Lista, som författare, en eller flera personer eller juridiska personer som ansvarat för förändringarna i den ändrade versionen, tillsammans med minst fem av de huvudsakliga författarna av dokumentet (alla dess huvudsakliga författare, om det har mindre än fem), såvida de inte ger dig tillstånd att bortse från detta krav.

C. Ange namnet på utgivaren av den ändrade versionen, som utgivare, på titelsidan.

D. Bibehålla dokumentets alla upphovsrättsklausuler.

E. Lägga till en upphovsrättsklausul för dina förändringar angränsande till de andra upphovsrättsklausulerna.

F. Direkt efter upphovsrättsklausulerna innefatta ett meddelande som ger allmänheten tillstånd att använda den ändrade versionen under villkoren i denna licens [det engelska originalet] i den form som visas i Tillägg nedan.

G. I meddelandet om licensen bevara den fullständiga listan över oföränderliga avsnitt och obligatoriska omslagstexter som finns i dokumentets meddelande om licensen.

H. Inkludera en oförändrad kopia av denna licens [Det är den engelska originalversionen som avses].

I. Bevara avsnittet med titeln "historik (History)", bevara dess titel och lägg i avsnittet till en post med åtminstone titeln, året, nya författare och utgivaren av den modifierade versionen så som angivet på titelsidan. Om det inte finns något avsnitt med titeln "historik (History)" i dokumentet så skapa en med titeln, året, författare och utgivaren av dokumentet så som det står på [original]dokumentets titelsida. Lägg sedan till en post som beskriver den förändrade versionen så som beskrivits ovan.

J. Bevara den nätverksadress, om det finns någon, angiven i dokumentet till den allmänt tillgängliga transparenta kopian av dokumentet, och likaså nätverksadresserna till de föregående versioner som dokumentet baseras på. Dessa får placeras i avsnittet "historik (History)". Du äger utelämna en nätverksadress för ett verk som är publicerat mer än fyra år före dokumentet självt, eller om den ursprunglige utgivaren vars verk nätverksadressen hänvisar till ger sitt tillstånd.

K. För alla avsnitt med titlarna "tillkännagivanden (Acknowledgements)" eller "dedikationer (Dedications)", bevara titeln på avsnittet, och bevara allt innehåll och prägel på alla tillkännagivanden och/eller dedikationer gjorda av varje bidragare.

L. Bevara alla oföränderliga avsnitt i dokumentet oförändrade till text och titel. Avsnittsnummer eller motsvarande anses inte tillhöra avsnittets titel.

M. Radera varje avsnitt med titeln "endossering (Endorsements)". Ett sådant avsnitt får inte inkluderas i en modifierad version.

N. Inte byta titel på något existerande avsnitt så att det blir "endossering (Endorsements)" eller så att titeln kan förväxlas med något oföränderligt avsnitt.

O. Bevara varje garantiavsägelseklausul.

Om den förändrade versionen innehåller nya framsidestexter eller bilagor som är att anses som sekundära avsnitt och inte innehåller något material kopierat från dokumentet, så äger du, om du vill, benämna några eller samtliga av dessa som oföränderliga. För att göra detta, lägg deras titlar till listan över oföränderliga avsnitt i den förändrade versionens licensmeddelande. Dessa titlar måste vara skilda från alla andra avsnitts titlar.

Du äger lägga till ett avsnitt med titeln "endossering (Endorsements)", förutsatt att det inte innehåller något annat än endosseringar för din modifierade version från olika aktörer -- till exempel, meddelanden om utförd korrekturläsning eller att texten har godkänts av en organisation som en officiell definition av en standard.

Du äger lägga till ett textavsnitt på upp till fem ord som framsidestext, och ett textavsnitt på upp till 25 ord som baksidestext i listan över omslagstexter i den modifierade versionen. Bara ett textavsnitt med framsidestexter och ett med baksidestexter får läggas till av (eller genom försorg av) en enda juridisk person. Om dokumentet redan innehåller en omlagstext för något av omslagen, tidigare tillagd av dig eller genom försorg av samma juridiska person som du företräder, äger du inte lägga till en till, men du äger ändra den gamla med tillstånd från den tidigare utgivaren som lade till den förra.

Författaren (författarna) och utgivaren (utgivarna) av dokumentet ger inte via denna licens sitt tillstånd att använda sina namn för publicitet eller för att lägga till eller antyda endossering av någon modifierad version.

D.6. 5. KOMBINERA DOKUMENT

Du äger kombinera dokumentet med andra dokument som är utgivna under denna licens, under de villkor som definieras i paragraf 4 av GNU Free Documentation License för modifierade versioner, förutsatt att du, i det kombinerade dokumentet, innefattar alla oföränderliga avsnitt från originaldokumenten, omodifierade, och listar dem som oföränderliga avsnitt i ditt kombinerade verk i dess licensklausul, och att du bevarar alla deras garantiavsägelseklausuler.

Det kombinerade verket behöver bara innehålla en enstaka kopia av denna licens [engelska originalversionen], och flera identiska oföränderliga stycken kan ersättas med en kopia. Om det finns flera oföränderliga stycken med samma namn men olika innehåll, se till att titeln på varje sådant avsnitt är unik genom att i slutet på den, inom parentes, lägga till namnet på den ursprunglige författaren eller utgivaren av det avsnittet om dessa är kända, annars ett unikt nummer. Gör samma justeringar av titlarna i listan över oföränderliga avsnitt i licensklausulen i det kombinerade verket.

I det kombinerade verket måste du kombinera alla avsnitt med titlarna "historik (History)" i de ursprungliga dokumenten, till ett avsnitt med titeln "historik (History)"; på samma sätt skall alla avsnitt med titlarna "tillkännagivanden (Acknowledgements)", alla avsnitt med titlarna "dedikationer (Dedications)" kombineras. Du måste ta bort alla avsnitt med titlarna "endossering (Endorsements)".

D.7. 6. SAMLINGAR AV DOKUMENT

Du äger skapa en samling bestående av dokumentet och andra dokument som är släppta under GNU Free Documentation License, och ersätta individuella kopior i dokumenten av denna licens med en enda kopia [av den engelska originalversionen] som inkluderas i samlingen, förutsatt att du följer villkoren för ordagrann kopiering i denna licens för varje inkluderat dokument i alla andra avseenden.

Du äger lyfta ut ett dokument från en sådan samling, och sprida det enskilt under GNU Free Documentation License, förutsatt att du lägger till en kopia av denna licens [den engelska originalversionen] i det utlyfta dokumentet, och följer villkoren för ordagrann kopiering i denna licens för det utlyfta dokumentet i alla andra avseenden.

D.8. 7. SAMMANSLAGNING MED OBEROENDE VERK

En samling av dokumentet eller av dess derivat med andra separata och oberoende dokument eller verk, på eller i en lagringsvolym eller ett spridningsmedium, kallas för en "sammanslagning" om den sammanslagna upphovsrätten inte används för att begränsa samlingens användares rättigheter som de enskilda dokumenten medger. När dokumentet ingår i en sådan sammanslagning, gäller inte denna licens de andra verken i samlingen som inte själva är deriverat av dokumentet.

Om kravet på omslagstexter enligt paragraf 3 är tillämpligt på dessa kopior av dokumentet, så kan dokumentets omslagstexter, om dokumentet utgör mindre än hälften av hela samlingen, placeras på det omslag som omger dokumentet inuti samlingen, eller den elektroniska motsvarigheten till omslag om dokumentet är i elektronisk form. Annars måste de synas på det omslag som omger hela samlingen.

D.9. 8. ÖVERSÄTTNING

Översättning anses vara en sorts förändring, så du äger sprida översättningar av dokumentet enligt de villkor som sätts i paragraf 4. Oföränderliga avsnitt som ersätts med översättningar kräver tillstånd från deras upphovsrättsinnehavare, men du äger inkludera översättningar av alla eller vissa av dessa oföränderliga avsnitt tillsammans med originalversionerna av dessa oföränderliga avsnitt. Du äger inkludera en översättning av denna licens, och alla licensklausuler i dokumentet, och alla garantiavsägelser, förutsatt att du också innefattar den engelska originalversionen av denna licens och originalversionerna av dessa klausuler. Skulle det finnas skillnader mellan översättningen och originalversionen av denna licens eller någon klausul så gäller originalversionen.

Om ett avsnitt i dokumentet har titeln "tillkännagivanden (Acknowledgements)", "dedikationer (Dedications)", eller "historik (History)", kommer kravet (paragraf 4) att bevara dess titel (paragraf 1) vanligtvis att kräva att själva titeln ändras.

D.10. 9. UPPHÖRANDE

Du äger inte kopiera, förändra, omlicensiera eller sprida dokumentet annat än enligt villkoren i GNU Free Documentation License. Alla övriga försök att kopiera, modifiera, omlicensiera, eller sprida dokumentet är ogiltiga och kommer automatiskt medföra att du förlorar dina rättigheter enligt denna licens. Tredje man som har mottagit kopior eller rättigheter från dig enligt dessa licensvillkor kommer dock inte att förlora sina rättigheter så länge de följer licensvillkoren.

D.11. 10. FRAMTIDA VERSIONER AV DENNA LICENS

Free Software Foundation kan publicera nya, reviderade versioner av GNU Free Documentation License då och då. Sådana nya versioner kommer att vara likadana i andemening som den nuvarande versionen, men kan skilja i detalj för att behandla nya problem eller angelägenheter. Se http://www.gnu.org/copyleft/.

Varje version av licensen ges ett unikt versionsnummer. Om dokumentet stadgar att en specifik numrerad version av denna licens "eller valfri senare version" gäller för det, så äger du rätten att följa villkoren enligt antingen den angivna versionen eller vilken senare version som helst som publicerats (inte som utkast) av Free Software Foundation. Om dokumentet inte anger en version av denna licens, äger du välja vilken version som helst som publicerats (inte som utkast) av Free Software Foundation.

D.12. TILLÄGG: Hur du använder denna licens för dina dokument

För att använda GNU Free Documentation License för ett dokument du har skrivit, inkludera en kopia av licensen [det engelska originalet] i dokumentet och placera följande copyrightklausul omedelbart efter titelsidan:

> Copyright (c) ÅRTAL DITT NAMN. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".
>
> [Översättning:] Copyright (c) ÅRTAL DITT NAMN. Var och en äger rätt att kopiera, sprida och/eller förändra detta dokument under villkoren i licensen "GNU Free Documentation License", version 1.2 eller senare publicerad av Free Software Foundation, utan oföränderliga avsnitt, utan framsidestexter och utan baksidestexter. En kopia av denna licens finns med i avsnittet med titeln "GNU Free Documentation License".

Om du har oföränderliga avsnitt, framsidestexter och baksidestexter, ersätt "with...Texts." ("utan...texter.") med följande:

> with the Invariant Sections being LISTA DERAS TITLAR, with the Front-Cover Texts being LISTA, and with the Back-Cover Texts being LISTA.
>
> [Översättning:] med de oföränderliga avsnitten LISTA DERAS TITLAR, med framsidestexterna LISTA, och med baksidestexterna LISTA.

Om du har oföränderliga avsnitt utan omslagstexter, eller någon annan kombination av de tre, slå samman dessa två alternativ så att det passar den uppkomna situationen.

Om ditt dokument innehåller icke-triviala exempel med programkod, så rekommenderar vi att du släpper dessa exempel parallellt under en, av dig vald, fri programvarulicens, som till exempel GNU General Public License, för att möjliggöra deras användning i fri programvara.