Programmeren 1

Planning: (programma van toetsing en afsluiting OER)

Individueel

We hebben 8 werkbare weken voor de 28+ opdrachten dus probeer er 4 per week af te ronden.

Programmacode is net zo individueel als een handschrift dus maak ze zelf!

Programma's moeten worden bewaard op schijf en op verzoek van de docent steekproefgewijs werkend getoond worden. (zorg voor backups in je mailbox)

Het programmaontwerp en de programmacode moeten worden geprint en in een portfolio worden verzameld.

Vervangende opdracht: (voor leerlingen die echt de applicatieontwerp of HBO kant op willen en door het docententeam op basis van resultaten en motivatie hiertoe in staat geacht worden)
Leerlingen die meewerken aan het onsbord project kunnen het vak programmeren afronden in deze context.
Daartoe moet een werkend programma worden gemaakt dat werkt als smoothboard / power presenter. Groepjes van 2 werken in onderlinge competitie. (dit is een contractopdracht, dus niet vrijblijvend, een onvoldoende is geen optie)

Herkansing:
Als blijkt dat de (vervangende)opdrachten niet op tijd voldoende worden afgerond dan volgt er een eindtoets over de theorie en een programmeeropdracht die op de toetsdatum moet worden gemaakt!

Deze reader wordt gepubliceerd op http://www.goudappel.org/onderwijs

Programmeren.

De vaardigheden die een programmeur bezit zijn onafhankelijk van de taal waarin hij/ zij werkt. De taal is een keuze die gemaakt wordt nadat het programma ontworpen is. Leren programmeren is zoiets als leren autorijden, op je rijbewijs staat niet in welk merk of model auto je moet rijden.
Het ontwerpen van een programma begint met het precies beschrijven van het probleem en mogelijke oplossingen. Probeer dit zo systematisch mogelijk te doen. Dit verhaal is de basis van het latere programma, en kan als commentaar deel blijven uitmaken van de code.
In de systematische beschrijving kun je heel goed gebruik maken van de zogenaamde PSD's (programmastroomdiagrammen of flowcharts)
Een verkenning op dit gebied is roboprog, waar de flowchart leidt tot een werkend programmaatje.

Verschillende programeertalen hebben allemaal hun eigen kwaliteiten.
Voor MBO-studenten zijn er een paar duidelijk het meest geschikt, ofwel omdat ze een breed toepassingsgebied hebben in het werkveld ofwel omdat ze dwingen tot goed programmeren.
Bovenaan staat **PERL** In een beheersomgeving is dit naast gewone **shellscripts** het meest toegepaste gereedschap. Vrijwel alle servers en services worden hiermee geconfigureerd. Daarnaast kan het als **CGI**-taal worden gebruikt om webapplicaties te ontwikkelen voorbij de mogelijkheden van PHP. Ook de interactie met databases en tekstbestanden / spreadsheets is met PERl uitstekend geregeld.
PHP is natuurlijk de taal om serversided HTML / XML in te genereren.
JavaScript is de enige mogelijkheid om programma's in een browser te laten lopen. JavaScript is door de opkomst van **AJAX** en Web 2.0 applicaties weer helemaal in de picture. (Google.docs, Google.search, Google.mail, enz)
Visual Basic VB wordt gebruikt om de macros in word en excel te maken.
JAVA is bij uitstek geschikt om kleine zelfstandige applicaties met een grafische interface te ontwikkelen omdat die op elk platform lopen.
Pascal is bij uitstek een taal die leert om goed te programmeren.
C en C++ zijn voor ons van minder belang niet omdat de taal zo moeilijk is, maar omdat de applicaties waarvoor het het meest geschikt is doorgaans heel ingewikkeld zijn, meer iets voor een HBO-er.
C# en andere .NET toepassingen zijn de zoveelste onsympathieke poging van Microsnot om een eigen platformgebonden dialect te introduceren van bestaande programmeeromgevingen en kan samen met ASP, Silverlight en Jscript in de overbodige zooi bak.

Variabelen, Constanten, Functies, datatypes.

Enkelvoudige datatypen: en hun ruimtegebruik

Name	Description	Size*	Range*
char	ASCII	1byte	unsigned: 0 to 255
char	Unicode	4 bytes	unsigned: 0 to 4294967295
int / byte	Tiny integer	1 byte	signed: -128 to 128 unsigned: 0 to 255
int	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value. It can take one of two values: true or false.	1byte	true or false
float	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)

Tekst types (ascii, unicode) (letters, woorden, tekstblokken, documenten)
Getaltypes (real, int, byte (unsigned))
Getallen zijn er in soorten en maten, we onderscheiden bijvoorbeeld scalairen(1), vectoren/pointers(2), coördinaten(2 of meer) afhankelijk van het aantal dimensies waarin een waarde zich afspeelt.
Logisch type (true, false)

Datum en tijd types (dd-mm-jjjj hh:mm:ss) zijn text, int of real

01-01-09 39814 donderdag 1 januari 2009 01-01-2009 00:00:00 955536:00:00,00 Tue, 1 Jan 2009 00:00:00 +0100 GMT

Constanten zijn 'variabelen' met een vaste waarde, kosten minder geheugen dan de waarde zelf, omdat ze worden hergebruikt, bovendien zijn ze sneller toegankelijk omdat ze al in het geheugen aanwezig zijn.
verhogen ook de leesbaarheid van programma's
pi() en exp() zijn voorbeelden

Complexe data types: array, hash, record, file, set, enum.

Arrays kun je je voorstellen als horizontale rijen de elementen worden benaderd door hun nummer:

@array(Waarde 1, Waarde 2, Waarde 3, Waarde 4, Waarde 5, Waarde n)

$array[1] heeft de waarde: Waarde 2 (begint bij 0) (en is een scalair)

Om het aantal elementen te tellen kun je een array in scalaire context aanroepen
$elementen = @array
$elementen = scalar(@array)

een array kun je gewoon weergeven met print @array

Hashes lijken meer op kolommen, elementen van een hash hebben een naam en een waarde, die waarde kan ook een referentie of zelfs een heel prgramma(blok) of subroutine / functie zijn

```
%hash(Sleutel1 =>  Waarde 1)
%hash(Sleutel2 =>  Waarde 2)
%hash(Sleutel3 =>  Waarde 3)
%hash(Sleutel4 =>  Waarde 4)
%hash(Sleutel5 =>  Waarde 5)
%hash(Sleuteln =>  Waarde n)
```

$hash{Sleutel4} heeft de waarde: Waarde4 (en is een scalair, referentie of programma(blok))

Om het aantal elementen te tellen kun je een hash in scalaire context aanroepen
$*aantal* = (*keys* %*hash*)
$*aantal* = (value*s* %*hash*)

een hash kun je afdrukken door:
while (($key,$value) = each %hash) {print "$key => $value\n";}

Hashes zijn buitengewoon efficient in hun ruimtegebruik, leesbaarheid en snelheid. Ook het zogenaamde verhashen van variabelen maakt ze geschikt voor het versleutelen van wachtwoorden en dus beveiliging.

Hashes en arrays kunnen ook weer hashes of arrays bevatten en zodoende als meer-dimensionale structuren worden benut.

Een **set** is een deelverzameling
set (Maandag,Dinsdag,Woensdag,Donderdag,Vrijdag,Zaterdag,Zondag)
set (A,Bb,B,C,C#,D,Eb,E,F,F#,G,G#)
in bijvoorbeeld C++ en MySQL heet dit een ENUM
ENUM('small', 'medium', 'large')
Het zijn eigenlijk arrays, die weinig ruimte innemen. (lagere kardinaliteit)

record
Te vergelijken met een kaartsysteem, in een record zitten variabelen van verschillende types

file
Te vergelijken met een buffer, heeft dezelfde structuur als een file op schijf, maar dan in het geheugen.

Kies zinvolle variabelen-namen
Herkenbaarheid variabelen $, @, %
de scoop van varabelen, global,local (my $variabele)

Niet alle talen eisen dat je varabelen van tevoren declareert, toch zou je dat altijd moeten doen, het voorkomt veel problemen achteraf.

Niet toegestane variabelenamen : spaties, verboden tekens, cijfers als eerste, taalelementen. In de meeste serieuze talen wordt onderscheid gemaakt tussen hoofd en kleine letters!

Operatoren (+, -, *, /, div, mod, ^ of **,++, --)

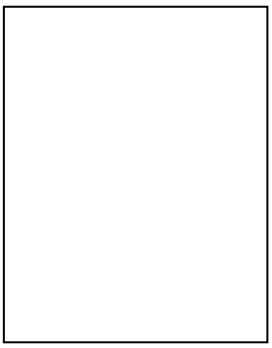
Operatoren zijn gekoppeld aan datatypes, bijvoorbeeld bij teksten betekent + dat de strings aan elkaar geregen worden (concateren), bij integers en reals is het een optelling.
Bij booleans leiden deze operatoren tot een foutmelding.
Intergers delen kan met div en mod het resultaat is dan ook een integer.
Reals delen doe je met / als je intergers deelt met een / dan is het resultaat een real.
Dat lijkt raar, maar stel je maar eens voor dat je datums van elkaar aftrekt, het resultaat is dan een aantal dagen en niet een nieuwe datum
binaire getallen kun je ook nog shiften naar links shiften komt overeen met 2x

Vergelijking, toekenning, verwerkingsvolgorde
= of := (soms met LET) toekenning (wordt of moet worden) of == vergelijking (is gelijk aan ?)
Andere vergelijkingsoperatoren zijn: > < => =< AND OR || && != NOT

Niet elke compiler / interpreter past automatisch meneer van Dalen toe. Bovendien gelden bij logische operatoren zelfs nog verschillende regels voor && en AND. Werk veel met haakjes om de juiste volgorde af te dwingen, dat geeft ook structuur aan je algoritmen. Er zit wel een beetje meneer van Dalen in, maar er wordt van links naar rechts gewerkt
Bv: 16:2x4 = 16:8 = 2 (want eerst vermenigvuldigen en dan pas delen)
Wordt op de computer: 16/2*4 = 8*4 = 32 (van links naar rechts afwerken)
Wil je dat goed doen dan moet de formule als volgt:
16/(2*4) = 16/8 = 2

Hoeken worden door de computer in radialen berekend, de belangrijkste reden daarvoor is dat dit decimale getallen zijn.
Het is altijd mogelijk om radialen en graden in elkaar om te rekenen
Graden = Radialen*180/pi
Radialen = Graden*180/pi

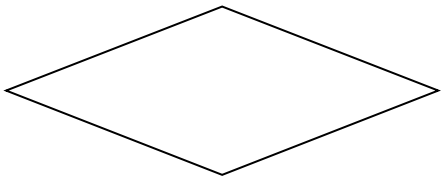
Programma blokken (classes)
Een blok is een samenhangend deel van een programma, het heeft in principe 1 uitgang en 1 ingang. Het programma zelf is een blok (tussen begin en end) en binnen een blok kunnen ook weer blokken voorkomen.
Je kunt een blok een naam geven, maar dat hoeft niet altijd.



Bijzondere blokken zijn:
Invoer (bv. toetsenbord, muis, bestand, modem/netwerk) alleen een uitgang
Uitvoer (bv. beeldscherm, printer, bestand) alleen een ingang
Acties die via buffers verlopen (netwerk toegang, schijftoegang) alleen de buffer, het besturingssysteem handelt de echte toegang af.

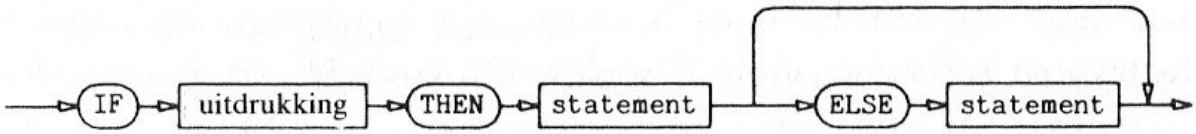
Beslissingstructuren: blokken die splitsen 1 ingang en ten minste 2 uitgangen. Bij een beslissingsstructuur zijn er ten minste twee mogelijke vervolgstappen, afhankelijk van een logische voorwaarde.

If then else
Elsif
Case

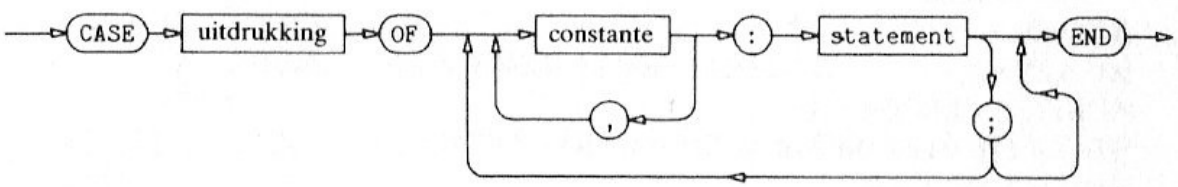
Nee  Ja

Een if then beslissing zit als volgt in elkaar:

if ($var != ''){doe dit} else {doe dat}



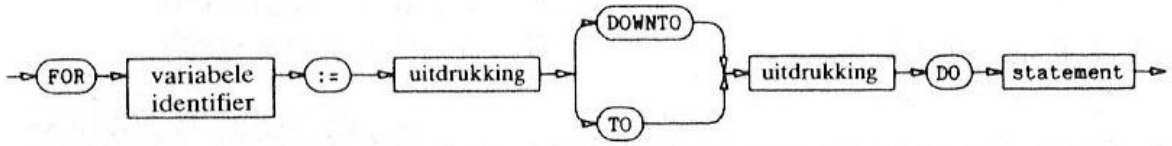
Een case beslissing zit als volgt in elkaar:



OO draw heeft alle onderdelen van organogrammen aan boord en autovormen/stroomdiagram in word

Lussen / iteraties: blokken die herhalen
Een lus is een stukje herhalende code, aan het begin of aan het einde wordt geëvalueerd of de lus herhaald wordt of dat eruit gesprongen moet worden.

Er zijn drie types lussen:
Lussen met een teller / lussen die aan stapel afwerken (for(-next), foreach, each)

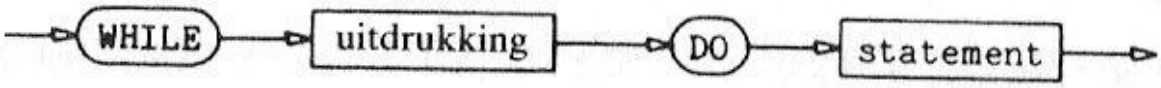


for ($var=1;$var<6;$var++){doe dit}

voor arrays; foreach $value(@array){doe dit}

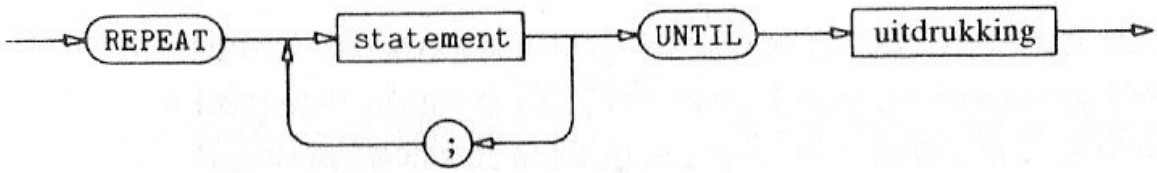
voor hashes; while (($key, $value) = **each** %contact) { doe dit;}

Lussen met de evaluatie vooraf (while(-wend of do))



while($var<=10){doe dit }

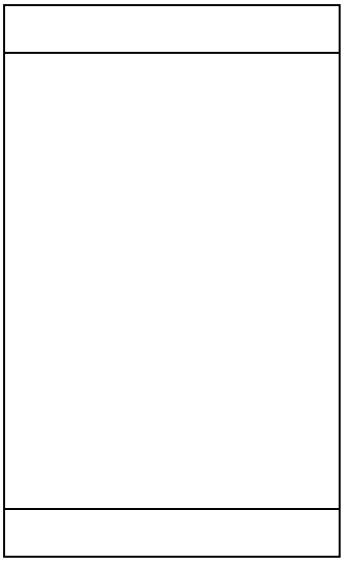
Lussen met de evaluatie achteraf (repeat until)



do {doe dit}while (*$var<=10*);

Veel programmeer problemen ontstaan als een lus blijft hangen, de voorwaarde wordt nooit waar.

Lussen kun je 'nesten' een lus in een lus maken.



Subroutines
Blokken met een naam die binnen een programma die kunnen worden hergebuikt.

Functies
Blokken met een naam die een 1 op 1 relatie leggen tussen input en output. veelgebruikte standaardfuncties zijn sin(), cos(), round() enzovoort. Functies behoren zelf ook tot een datatype.

Procedures en objecten
Blokken met een naam die in verschillende programma's kunnen worden gebruikt door ze in een bibliotheek te zetten.
Je kent het concept object al uit HTML waar je bijvoorbeeld het object <img> hebt. In dit object zit het plaatje, een kader, een alt tekst, enzovoort. Een ander object is de <a> waarin een link zit (href) en een target of mouseovers.
Een procedure kan ook een functie zijn.

Object georiënteerd programmeren (LEGO-doos).

Wat zijn de verschillen tussen de verschillende programmeertalen?
1 de syntaxis
2 de ingebouwde functies en procedures
3 de uitgebouwde functies en procedures (bv javascript kan niet naar een schijf schrijven en mag dat ook niet kunnen)
4 eisen aan variabelen (declaraties, typevastheid)
5 mogelijke complexe variabelen (hashes, records, sets)

Er zijn geïnterpreteerde en gecompileerde talen al dan niet op een virtuele machine.
Een just in time compiler (JIT) probeert de voordelen van beide te combineren.
Elke taal heeft zo zijn sterke (en zwakke) kanten.

Compilertalen: (hebben als nadeel dat ze voor elke soort computer/besturingssysteem (platform) apart moeten worden gecompileerd)
C (levert hele snelle programma's zoals de kernel van Linux, erg technisch)
C++ (objectgeoriënteerd, snelle programma's, erg technisch)
Pascal (strak en eenvoudig, geen objecten, dwingt tot goed programmeren)
Delphi (als Pascal maar dan met objecten, kan gebruikt worden om DLL's te maken)
Wat oudere talen zoals Fortran, Cobol, Algol

Scripttalen: (hebben interpreters nodig)
(Quick) Basic (weinig overhead, snel en slordig programmeren mogelijk)
Visual Basic (grafisch georiënteerd, is ook de macro taal van Windows applicaties)
JavaSript (samen met AJAX, ingebed in HTML, de interpreter draait in de browser)
PHP (speciaal voor webtoepassingen, de interpreter draait op de server)
Python (objectmatig, dwingt tot goed programmeren)
PERL (tekst bestanden en databases verwerken samen met een webserver, of onder Linux)
Maar ook de Shell talen van DOS en Linux

Compilertaal op een virtuele machine of een JIT:
Java (platformonafhankelijk, objectgeoriënteerd, sterk grafisch)
C# (platformonafhankelijk maar vooral Windows, gebruikt de .NET bibliotheken)

Moderne interpreters en JIT-compilers zijn zo snel dat kleinere programma's net zo snel zijn als gecompileerde talen. Scripttalen zijn altijd platform onafhankelijk, omdat de interpreter op het platform werkt.

Een script is er voor de regisseur, het programma is er voor de toeschouwer. (Larry Wall de ontwerper en coördinator van PERL)

"Programs must be written for people to read, and only incidentally for machines to execute." - Abelson & Sussman

C
```
main( ) {
  printf("hello, world");
}
```

C++

```
#include <iostream>
using name space std;

int main()
{
  cout << "Hello World" << endl;
  return 0;
}
```

C#

```
class HelloWorld {
        static void Main() {
                System.Console.WriteLine("Hello World");
        }
}
```

Pascal

```
Program HelloWorld(output);
Begin
 Writeln('Hello world');
End.
```

(Quick) Basic

```
10 PRINT "Hello World"
```

Visual Basic

```
MsgBox("Hello world!")
```

JavaScript

```
document.write('Hello world');
```

PHP

```
<?php
echo "Hello World";
?>
```

Python

```
print "Hello world"
```

PERL

```
print "Hello world\n";
```

Java

```
public class HelloWorld {
  public static void main(String args[]) {
    System.out.println("Hello world");
  }
}
```

Gestructureerd programmeren

Uit Wikipedia, de vrije encyclopedie

Gestructureerd programmeren kan worden gezien als een subdiscipline van procedureel programmeren, een van de belangrijke programmeerparadigma's (en waarschijnlijk de populairste) voor het programmeren van computers.

In de jaren '60 werd het langzaamaan duidelijk dat de complexiteit van computerprogramma's uit de hand begon te lopen. Programmeurs zagen door de bomen het bos niet meer.

Goede programmeurs wisten door structureel te werken echter code te produceren die veel makkelijker te doorzien was. Het viel op dat bekwame programmeurs met name veel minder GOTO-statements gebruikten. Wetenschappers in de informatica, waaronder met name Edsger Dijkstra, Jean-Dominique Warnier en Michael A. Jackson, zagen de oplossing in het gestructureerd programmeren, waarbij het door nieuwe constructies in programmeertalen onnodig zou worden constructies als het goto-statement te gebruiken, waardoor de leesbaarheid van code veel beter werd.

De drie basisconstructies in gestructureerd programmeren zijn:

- sequentie = opeenvolging (S1 S2)
- iteratie = herhaling (met de herhalingsvoorwaarde vooraf of achteraf: meestal while B do S en do S until B)
- selectie = keuzestructuur (if B then S1 else S2)

S staat voor een opdracht (Statement), B voor logische expressie (Boolean). Belangrijk is hierbij dat zowel een opdracht als een logische expressie willekeurig complex mogen zijn. In programmeertalen is dat voor opdrachten gerealiseerd door begin ... end-constructies:

- begin en end in Pascal
- de accolades { en } in Java, C en PHP

Men hoopte dat een en ander het bewijzen van de (wiskundige) correctheid van een programma mogelijk zou maken. (Dit is in de praktijk anders uitgevallen.)

Dit gestructureerde programmeren leidde tot een nieuwe generatie programmeertalen, waarvan met name de programmeertaal Pascal het ver geschopt heeft. Al deze talen kennen nog wel het goto-statement en het label, maar het gebruik ervan wordt sterk afgeraden.

enkelvoudige keuzestructuur

Bij een keuzestructuur gaan we een bepaald stuk code al of niet laten uitvoeren door de computer, afhankelijk van de voorwaarde.

structuur	beschrijving
if v then ...	als voorwaarde dan {...}
if v then ... else ...	anders voeren we een andere opdracht uit
if v then ...elseif...else	anders voeren we opnieuw een andere opdracht uit

voorbeeld van een if in PHP

```
<?
 $humeur = "slecht";
 if ($humeur == "slecht") {
    echo ("ik ben slecht gezind");
 }
 ?>
```

voorbeeld van een elseif in PHP

```
<?
 $humeur = "goed";
 if ($humeur == "slecht") {
    echo "ik ben slecht gezind";
 }
 elseif ($humeur == "verveeld") {
    echo "ik verveel mij";
 }
 else{
    echo "ik ben goed gezind";
 }
 ?>
```

meervoudige keuzestructuur

Zoals de naam al zegt, gaan we bij een meervoudige keuzestructuur een keuze maken ui meerdere mogelijkheden. deze zijn uiteraard ook te maken met een enkelvoudige, maar met een switch gaat dit makkelijker:

```
<?
 $humeur = "goed";
 switch ($humeur) {
 case "slecht":
   echo "ik ben slecht gezind";
   break;
 case "verveeld":
   echo "ik verveel mij";
   break;
 default:
   echo "ik ben goed gezind";
 ?>
```

voorwaardelijke herhalingsstructuur

Hier zijn 2 mogelijkheden:

- while: eerst testen, daarna de opdracht uitvoeren

PHP voorbeeld

```
<?
 $teller = 0;
 while ($teller <= 12) {
    echo "de teller zit nu aan: $teller <br />";
    $teller++;
 }
 ?>
```

- do while: eerst opdracht uitvoeren, dan pas testen

PHP voorbeeld

```
<?
 $teller = 0;
 do {
  $teller++;
  echo "de teller zit nu aan: $teller <br />";
  } while ($teller <= 12);
 ?>
```

begrensde herhalingsstructuur

Hierbij weten we op voorhand hoeveel maal we een bepaalde blok code willen herhalen:

```
<?
 for ($i = 1; $i <= 5; $i++) {
   echo "hallo nummer $i ! ";
 }
 ?>
```

Hier bevinden zich 3 parameters : 1) initialiseert de variabele 2) voorwaarde en 3) de actie die er moet gebeuren bij de herhaling.

Om een Array te doorlopen gebruiken we de foreach uitdrukking. Bij elke doorloop wordt de huidige waarde van de array gegeven aan de variabele $value tot de hele array doorlopen is:

```
<?
 $arr=array("een", "twee", "drie");
 foreach ($arr as $value) {
  echo "De waarde is $value <br />";
 }
 ?>
```

SDM

De SDM systematiek is ook van toepassing op het maken van programma's.
Dat wil zeggen dat je vooraf duidelijk moet vastleggen wat je wel en wat je niet gaat doen en wanneer dat klaar moet zijn. Ook kun je al van tevoren aangeven welke uitbreidingen er eventueel later kunnen worden toegevoegd. Verder wordt duidelijk wie verantwoordelijk is voor het aanleveren van gegevens en welke eisen daaraan gesteld worden.

1 Analyse, wat is het probleem?
2 Ontwerp, wat moet het programma doen? (functioneel)
Hoe moet het programma dat doen? (technisch, algoritmen)
3 Realisatie, het schrijven van de code, testen

Modulair werken.
Een goede programmeur is lui, vréése lijk lui. Niet op korte termijn maar in de loop van zijn/haar carrière. Dat betekent dat elke handeling die meer dan eens moet worden uitgevoerd vroeg of laat tot een programma leidt, maar ook dat veel programma's bestaan uit het knippen en plakken of linken van eerder gemaakte en geteste code.
Juist daarom is het van belang om goed te documenteren.
Stukjes goed gedocumenteerde code steeds opnieuw gebruiken leidt tot minder fouten, de code werkt immers al. Het opbouwen van bibliotheken van stukjes programma die werken is het begin van modulair programmeren. Maar ook binnen een programma kun je modulair werken, een van de mogelijkheden is om binnen een blok indentatie (inspringen) te gebruiken waardoor je duidelijk kunt zien wat bij elkaar hoort.

Algoritmen
Algortimen zijn programmastructuren die gegevens verwerken.
Aan algoritmen worden hoge eisen gesteld
1 Zo moet er van tevoren gedetaileerd vaststaan wat de invoer en uitvoervariabelen zijn en tot welk type ze behoren. vooral de invoer moet goed gecontroleerd worden. Gebruikers zijn onvoorspelbaar en onbetrouwbaar, maar ook bestanden kunnen fouten bevatten, om nog maar te zwijgen van beschadigde of verminkte bestanden.
2 Algoritmen moeten robuust zijn, ze moeten door foute invoer niet in de war raken, maar melden dat er iets fout is en niet proberen iets met de foute gegevens te doen.
3 Algoritmen moeten uitvoerbaar zijn, dat klinkt logisch, maar je komt regelmatig gevallen tegen (bijvoorbeeld het laden van een dataset in een database) dat iets niet lukt, het duurt te lang, het kost teveel geheugen of schijfruimte. stacks en swapfiles lopen over. (sorteren is een berucht voorbeeld)
4 Algoritmen moeten eenduidig en ondubbelzinnig zijn.
5 Algoritmen moeten eindig zijn.
6 Algoritmen moeten correct zijn, lijkt logisch maar is moeilijk te bereiken.
7 Algoritmen moeten efficient zijn.
8 Algoritmen moeten algemeen zijn. Dit bevorderd ook het hergebruik bijvoorbeeld in een bibliotheek of als object of procedure.

en zelfs als aan alle voorwaarden is voldaan kun je nog een GiGo systeem gemaakt hebben....

Het testen van programma's

IEEE-norm voor het testen van hard en software: IEEE 189-1998

Dus: wat moet er in een tesplan staan?
Een antwoord op de volgende vragen;

1 Uniek nummer-naam (Zodat je het terug kunt vinden.)
2 Wat wordt er getest? (Bijvoorbeeld: Software)
3 Welke onderdelen? (Bijvoorbeeld: een module)
4 Welke onderdelen worden niet getest? (Bijvoorbeeld: een subroutine)
5 Hoe? (Bijvoorbeeld: 5x met verschillende invoer)
6 Wat is goedgekeurd, wat is afgekeurd? (3 uit 5 of 100%)
7 Hoe vaak? (Bijvoorbeeld: 5x, 10x, totdat er iets fout gaat, totdat het drie keer achter elkaar goed gaat...)
8 Wat komt er minimaal in het rapport? (Bijvoorbeeld: sticker met approved en handtekening, of uitgebeid)
9 Wat ga je doen? (Bijvoorbeeld: testbestanden als invoer)
10 Wat heb je extra nodig? (Bijvoorbeeld: OS,compiler,interpreter)
11 Wie is verantwoordelijk?
12 Wie doet het werk?
13 Wanneer? (Bijvoorbeeld: Voor je verdergaat, na voltooiing van de gehele code, als de klant klaagt)
14 Risico´s en mogelijke vervuiling-besmetting. (Bijvoorbeeld: variabelen die niet zouden mogen worden geaccepteerd)
15 Verbeteringen. (Wat zou je de volgende keer anders willen doen?)

Het hele document in Engels; http://www.cs.ucf.edu/~workman/cen5016/IEEE829.pdf

Invoercontrole / validatie
datatype controleren
String → value (punt of komma als decimaal scheidingsteken, :, -, / in datums, postcodes, huisnummers)
Tainted data
Die en Warn

Foutzoeken
Syntax fouten (hoe is een zin / commando geschreven)
Semantiek fouten (wat betekent een zin / commando)
Compiler fouten (zijn vooral syntax fouten)
Runtime fouten (zijn vooral semantiek fouten)

Bijvoorbeeld while $variabele == $variabele do …
Is altijd waar, en dus zinloos, het levert runtime een fout op terwijl de syntaxis juist is, en de compiler er dus geen problemen mee heeft.

Dit heet in de taal een zinledige volzin, de zin klopt, maar slaat nergens op.

Markuptalen (HTML, XML) zijn geen programmeertalen, maar veel van de regels die voor het programmeren gelden zijn ook van toepassing op de markuptalen.

Reguliere expressies en SQL zijn talen die binnen andere talen gebruikt kunnen worden.

Interactie met bestanden
In het algemeen kunnen programma's niet met de bestanden zelf werken
Een bestand wordt geopend in een buffer in het geheugen die door het programma kan worden gelezen of veranderd.
vaak is het sneller en handiger om de buffer in variabelen op te slaan en dan te bewerken:

```
open(buffer,"<","bestand") or die "can't read betand $!\n";
@array = <buffer>;close(buffer);
```

Als het programma klaar is wordt de buffer weer naar de schijf geschreven.
je hebt hier als windows gebruiker ervaring mee, want als een programma crasht ben je de vernaderingen kwijt, die zijn ook gecrasht.
interactie met printers en soms met poorten of zelfs een netwerk lopen ook via buffers.
Al bij het openen van een bestand moet je aangeven wat er moet gebeuren, alleen lezen, veranderen, toevoegen.

Opdrachten algemeen:

Omdat programmeren zelf taalonafhankelijk is wordt er onderscheid gemaakt tussen programmeren en coderen.

De opdrachten bestaan uit het programmeren en daarna het coderen in een taal naar keuze (soms meerdere, soms de keuze van de docent)

Het ontwerp staat als commentaar in de programmatekst!

Programmeren gaat prima in Word of OOwriter, coderen gaat beter in een editor, zonder opmaak.

Iedereen heeft op zijn/haar eigen computer de volgende editor en talen werkend aanwezig.
Crimson editor of Arachnophilia. Hierin zitten templates voor gangbare talen en tag-coloring

PERL (zit in XAMPP)
PHP (zit in XAMPP)
Pascal 3
JAVA
Visual Basic
En uiteraard de DOS CMD Shell of de Linux BASH Shell
C++, GCC, of iets anders is facultatief en werken onder Linux mag ook (in dat geval K-edit als editor).

Zorg dat je altijd weet waar de manuals en syntaxisdiagrammen van de talen op het net te vinden zijn.

http://www.freepascal.org/
http://www.microsoft.com/express/vb/

Het is handig om de compiler aan je zoekpad van windows toe te voegen, dat spaart veel ergernis.
Ik heb bijvoorbeeld een bat-bestand gemaakt met de volgende regel erin
path h:\xampp\perl\bin\
als ik dit run, dan hoef ik daarna niet meer te zoeken naar de perl interpreter en kan ik overal vandaan perl scripts draaien.
De shebang bovenin een PERL script wijst ook de weg naar de compiler.

```
#!/bin/sh
#!/usr/bin/perl -w
```

Je kunt beginnen met al je compilers en interpreters te testen door voor elke taal een “Hallo wereld” programma te maken.

Opdrachten:

1. Ontwerp en maak een programma dat de tafels van 1 tot 13 in een tabel zet.
2. Ontwerp en maak een programma dat de grootst gemene deler van twee ingevoerde getallen berekent. mag ook een max, min of kgv zijn
3. Ontwerp en maak een programma dat aangeeft of een ingevoerd getal even of oneven is. (moduuldeling en odd-functie)
4. Ontwerp en maak een programma dat de stelling van Pythagoras toepast
5. Als 2 maar nu moet ook worden gecontroleerd of de invoer klopt (geen tekens en letters).
6. Als 3 maar nu moet het programma aangeven wat er fout gegaan is.
7. Maak een command line rekenmachine die kan optellen vermenigvuldigen delen en aftrekken met drie invoer momenten.
8. Maak een command line rekenmachine die kan optellen vermenigvuldigen delen en aftrekken met één invoer moment. (parsing)
9. Maak een command line rekenmachine die kan optellen vermenigvuldigen delen en aftrekken met invoer als argumenten.
10. Als 5 of 6 of 7 maar nu moet in een subroutine worden gecontroleerd of de invoer klopt (alleen toegestane tekens en cijfers).
11. Ontwerp en maak een programma met een functie die Celcius in Fahrenheit omrekent en omgekeerd.
12. Ontwerp en maak een programma met een functie die het gemiddelde van een reeks cijfers uitrekent.
13. Als 5 of 6 of 7 maar nu ook met hexadecimale getallen.
14. Ontwerp en een programma dat van binair naar hexadecimaal omrekent ook moet worden gecontroleerd of de invoer klopt (alleen toegestane tekens en cijfers).
15. Als 1 maar nu met array's.
16. Ontwerp en maak een programma dat bingogetallen genereert en bijhoudt in een tabel (2D-array) welke getallen al geweest zijn.
17. Ontwerp en maak een programma dat pokerkaarten schudt en deelt (gebruik 4 arrays) en een willekeurige stack overhoudt.
18. Ontwerp en maak een programma dat door middel van hashes letters in morse code vertaalt.
19. Ontwerp en maak een programma dat een tekstbestand inleest en daarin het aantal woorden telt.
20. Ontwerp en maak een programma dat het aantal dagen uitrekent tussen 2 data. (maandlengte, schrikkeljaren) extra: in uren (DST!)
21. Ontwerp en maak een programma dat het spel mastermind speelt.
22. Ontwerp en maak een programma dat een (webserver)logfile ophaalt, een backup maakt, gegevens analyseert, opmaakt en in een nieuw (HTML)bestand wegschrijft.
23. Ontwerp en maak een programma ongeacht de extensie van een bestand aan de header zien kan wat voor type bestand het is (doc,gif,jpg,txt,mp3)
24. Ontwerp en maak een programma dat directorylistings (pad/filename.ext) maakt van een hele harddisk, opmaakt en in een nieuw (HTML)bestand wegschrijft. (eventueel met filesize, datum en attributen)
25. Bestand -> Sorteeralgoritme -> Bestand (niet sort gebruiken)
26. Gebruik binnen PHP het exec commando om programma 18 aan te roepen.

27. Gebruik 24 om alle bestanden die de afgelopen 2 uur gemaakt of veranderd zijn te vinden.
28. Ontwerp en maak twee programma's in twee talen de ene maakt een directory en een bestand en past de rechten aan. De andere haalt dit bestand op en maakt een backup (read only)

We hebben 8 werkbare weken voor deze 28+ opdrachten dus probeer er 4 per week af te ronden.