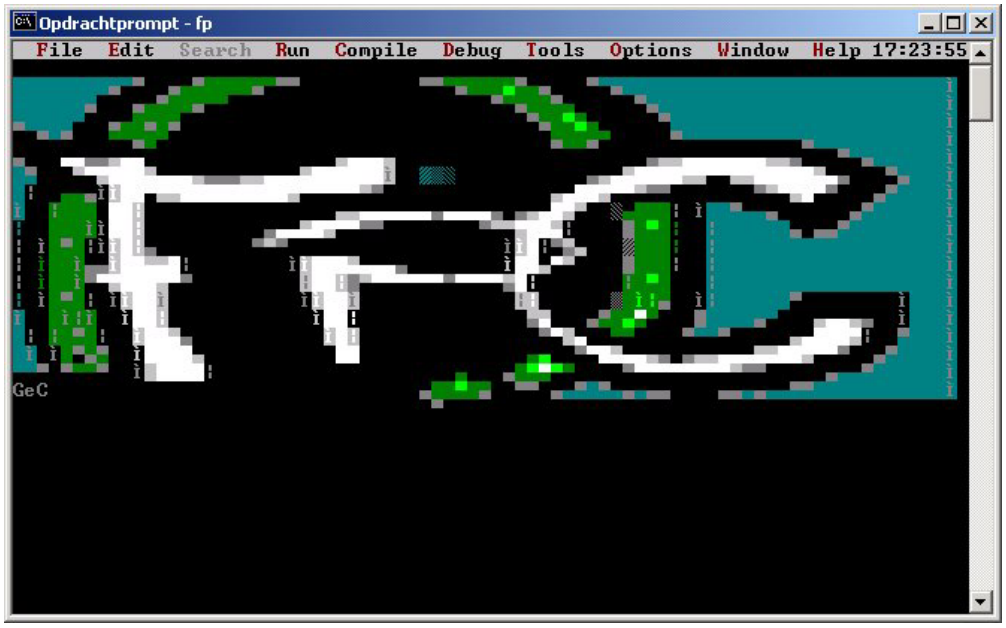
Voorwoord

Leren programmeren lijkt voor maar heel weinig mensen weggelegd.
Dit boekje wil dat tegenspreken.
Meestal zijn boeken en handleidingen voor programmeertalen ronduit saai. Geschreven door zeer deskundige mensen die het programmeervak waarschijnlijk tot in de toppen van hun tenen beheersen, bieden deze boeken meestal stof voor diegenen die zich al enigszins thuis voelen in de wereld van de natuurkunde, de bits, de bytes, geheugenadressen, berekeningen en variabelen..
Een beginnend programmeur wordt snel afgestoten door te veel theorie waar geen handen en voeten aanzitten. En zal dan dus nooit verder komen als:"beginnend programmeur"
Als dan ook de opgaven, die bij leerstof natuurlijk niet kunnen ontbreken, altijd bestaan uit 'hello world', saaie hypotheekberekeningen en databases met klantgegevens,dan is het niet verwonderlijk dat vooral jonge mensen hun interesse verliezen en afhaken.
"Donder maar op met die saaie zooi!!" Roept de zapgeneratie. En terecht.
Met een beetje fantasie kunnen dezelfde vaardigheden aangeleerd worden op een manier die wat meer aanspreekt.
Minder uiteenzetten met moeilijke woorden maar aanleren door toepassen.
Dit boekje is absoluut geen veelomvattende handleiding, laat staan een werk voor de gevorderde programmeur.
Deze handleiding is vooral bedacht om te enthousiastmeren en om de beginselen van turbo pascal bij te brengen.
Tot op niveau van dit boekje is alles makkelijk terug te vinden, en zijn de syntaxis van elke functie overzichtelijk terug te vinden via de inhoudsopgave achterin het boek.
De Plaatjes en tekeningen helpen ook om thuis te geraken in het boek. Het stimuleren van linker en rechter hersenhelft is een nieuwe manier om dingen sneller aan te leren. Dit heb ik hier proberen te bereiken door veel met taal, geluid en kleur **(linker hersenhelft)** te doen in plaats van alleen maar wiskundige berekeningen op een wiskundig/natuurkundige**(rechter hersenhelft)** manier uitleggen.

Freepascal.
De IDE die voor dit boekje gebruikt is, is de IDE van freepascal. Het is een download van ongeveer 20 mb en je kan aan de slag met een volledige ontwikkelomgeving voor turbo pascal.
Eenmaal geinstalleerd neemt het ongeveer 100 mb schijfruimte in beslag. Dat valt dus wel mee voor een serieuze programmeer omgeving. Downloaden bij

www.freepascal.org unzippen en dubbelklikken op install.exe. De installatie zal automatisch een map op de c- schijf maken en daar alle bestanden inzetten. Ga naar de map waar freepascal geinstalleerd is (standaard c:\pp) en open de map bin. Daarin zit de map win32 en daarin het bestand fp.exe. Dubbelklik fp.exe en je IDE start op.



Hoe ga je aan de slag met dit boek?
Zorg dat je een IDE hebt zoals bijvoorbeeld die van Free Pascal. Bekijk even hoe je een file opent en sluit, hoe je kan compilen en runnen. Gebruik hiervoor even een voorbeeldje uit het boek.
Als dat lukt, kan je beginnen met het eerste hoofdstuk. Lees eventjes een stukje totdat je bij een voorbeeld komt. Type dat over in je IDE en kijk of het werkt. Lees daarna het stukje nog een keer over, waarschijnlijk zal je dan begrijpen wat de code betekend die je net hebt overgetypt. Als dat zo is kan je de opgaven gaan maken. De opgaven zullen meestal bestaan uit een programma dat werkt met de zelfde code's die er in het voorbeeld staan, alleen moet er nu een beetje van je eigen creativiteit aan toegevoegd worden. Op deze manier kan je doorwerken tot het eind van het boek.

INHOUDSOPGAVE

Hoofdstuk 1
JE EERSTE PROGRAMMA MET BEELD EN GELUID

Een programma in turbo pascal bestaat **altijd minimaal** uit de volgende onderdelen.

1. Een programma naam.
2. een stukje waarin aangegeven wordt welke Units er gebruikt worden. (het begrip Unit wordt later behandeld).
3. Een begin.
4. Opdrachten.
5. Een eind.

Bv.

```
1.    PROGRAM U_KIJKT_NAAR_EEN_SCHERM;
2.    USES CRT;

3.    BEGIN
4.    Writeln('U kijkt naar een scherm.');
5.    END.
```

Even vertalen hoor:
In **regel 1** staat het woord PROGRAM gevolgd door een naam die geen spaties mag bevatten. De regel wordt afgesloten door een ; (puntkomma) zoals de meeste regels in een programma. Kijk voor de uitzonderingen naar de voorbeelden.
Regel 2 bestaat uit de woorden USES en CRT. Het Engelse USES staat voor 'gebruikt' en CRT staat voor de CRT-unit. De compiler zal dus de CRT-unit gaan gebruiken bij het compilen.
Over wat die Units precies inhouden gaan we het nu niet hebben. Voorlopig moet je deze code gewoon gebruiken in alle programma's die je schrijft.
Regel 3 BEGIN geeft aan dat hier het eigenlijke programma begint.
Regel 4 Writeln betekend:schrijf een regel. Daarna komt een text die tussen twee aanhalingstekens en dat staat weer tussen haakjes. De tekst wordt op deze manier letterlijk afgedrukt op het scherm.
Deze opdracht wordt weer afgesloten door een ; (puntkomma).

Regel 5 maakt duidelijk dat hier het einde van het programma is d.m.v. het woordje END en de . (punt) die daar weer achter staat.

Program
De programmeercode van een programma begint altijd met het woordje program, spatie, dan komt de naam van het programma waar letters, cijfers en underscores in mogen zitten. De naam van een programma wordt afgesloten met een ; (puntkomma).
Syntax:

```
Program dit_is_programma_deel_1;
```

Deze naam mag je verder niet in je programma gebruiken.

Uses
De code uses komt direkt na de eerste regel met de programma-naam. Achter uses zet je de namen van de units die de compiler moet gebruiken om je programma te compileren. Tussen de namen van de units zet je komma's na de laatst unit-naam zet je een ; (puntkomma).
Syntax

```
Uses Crt, Dos;
```

Een unit teveel zal niet zo veel uitmaken, een unit te weinig wel.
In units zit de uitleg van sommige opdrachten die de compiler zelf niet kent. Bv de opdracht ClrScr zit in de unit crt. Units komen later in dit schrijven nog uitgebreid aan de orde.

Writeln;
De functie writeln betekend; "schrijf de volgende regel"
Syntax:

```
Writeln('nou nou wat een text');
```

Je schrijft dus writeln en daarachter haakje openen. Om de tekst die je op het scherm wilt laten zien zet je apostrofs (' ') één voor en ook één achter de text. Nu zal die regel letterlijk op het scherm verschijnen.

Begin…End;
Deze worden gebruikt om, je kan het eigenlijk al raden, een begin en een eind van iets aan te geven. Dat iets kan meerdere dingen zijn.

1. Een programma begint met Begin en eindigt op End.. (jawel met een punt)

2. Een stukje programmatext dat bij elkaar hoort begint met Begin en eindigt op End;.
3. De programmacode van een procedure of een functie begint met Begin en eindigt op End;.

Begin schrijf je zonder punt of komma end altijd met een ; (puntkomma) en aan het eind van het hoofdprogramma altijd met een .(punt).

Syntax:

```
Begin
Hier komt allemaal programmacode
        Begin
        Hier zou een bij elkaar horend stukje code kunnen staan.
        End;
Hier komt nog wat programmeer-geneuzel.
End.
```

GotoXY

Het scherm waarin je je programma's uitvoert (dat zwarte) bestaat eigenlijk uit 80 * 25 vakjes. Ofwel 80 kolommen van 25 regels.
Met GotoXY kan je de cursor verplaatsen naar de x,y coördinaat die er na komt.
Syntax:

```
GotoXY(35,10);
```

Na deze aanroep zal de cursor zich dus 35 stapjes naar rechts en 10 stapjes naar beneden begeven en daar gaan zitten wachten op het volgende commando.

Bv.

```
Program het_midden;
Uses crt;
Begin
GotoXY(40,21);
Writeln('* dit is het midden');
End.
```

Let op:

Bij Gotoxy: x is van links naar rechts y is van boven naar beneden

Opdracht 1_a

Schrijf een programma dat de woorden GA WEG in de vier hoeken van je scherm laat zien.
Je moet dus even letters tellen!

Delay
Met de functie Delay kan je de computer vertellen dat hij even moet wachten. Dit kan je gebruiken als je je programma meerdere dingen na elkaar wilt laten doen.
Syntax:

Delay(500);

Na de opdracht Delay zet je een getal tussen haakjes dat aangeeft hoeveel milliseconden de pauze moet duren.

ClrScr
Zoals je misschien wel gezien hebt wordt het een beetje een zooitje op dat zwarte uitvoerscherm.
Dat kan opgeruimd worden met de opdracht ClrScr dat **Cl**ea**R SCR**een betekent.
Syntax:

ClrScr;
Na deze opdracht is alles weer netjes op dat zwarte scherm.

Readln I
Behalve dat je wat kan schrijven naar het scherm kan je er ook wat van aflezen met behulp van de functie Readln. De functie gaat zitten wachten tot je wat getypt hebt. Omdat een programma vaak nog iets op het scherm zet dat we graag willen zien, gebruiken we readln om het programma te laten wachten op het indrukken van een toets van het toetsenbord, voordat het zich afsluit.
Syntax:

Readln;
Netjes een puntkomma erna dus.

```
Program opruimen;
Uses crt;
begin
Writeln(`  zzzzzzzzzzzzzzzzzzzzzzzzzzzzz      ');
Writeln(`    ooooooooooooooooooooooooooo ');
```

```
Writeln(`      ooooooooooooooooooooooooooo     ');
Writeln(`     ooooooooooooooooooooooooooooo    ');
Writeln(`   ooooooooooooooooooooooooooooooo  ');
Writeln(`      oooooooooooooooooooooooooooooooooooooo   ');
Writeln(`    ttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttt  ');
Writeln(`   jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj     ');
Writeln(`    eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee       ');
Delay(500);
Writeln('WAT EEN ZOOI HE?');
Writeln('over 3 tellen is het weg');
Delay(500);
Writeln('nog 2');
Delay(500);
Writeln('nog 1');
Delay(500);
ClrScr;
Delay(500);
GotoXY(35,10);
Writeln(mooi he?);
Readln;
End.
```

Die laaste Readln, wacht dus op een teken (bijvoorbeeld een <enter>) totdat het programma afsluit. Probeer het maar eens weg te laten en kijk wat er gebeurt.

Opdr1_b;
Schrijf nu zelf een programma dat de text 'GA WEG' in de vier hoeken laat starten en langzaam naar het midden van het scherm 'loopt'.
Maak hier minimaal 5 stappen van.
Dus vijf keer writeln en 5 delay's even flink puzzelen met GotoXY.

Er kan ook met kleur gewerkt worden binnen turbo pascal.
De text en de achtergrondkleur kunnen veranderd worden. Maak ze nooit dezelfde kleur!
De kleuren zijn de volgende:

0 = zwart	8 = antraciet
1 = blauw	9 = lichtblauw
2 = groen	10 = lichtgroen
3 = blauwgroen	11 = blauwgroen
4 = rood	12 = lichtrood
5 = paars	13 = paars/roze
6 = bruin	14 = geel
7 = lichtgrijs	15 = wit

Textbackground
Met textbackground kan je de achtergrondkleur van de letters op je scherm bepalen.
Hierboven zie je welke getallen voor welke kleur staan.
Syntax:

```
Textbackground(1);
```

Nu krijg je dus een blauwe achtergrond achter de letters. Als je een mooi egaal scherm wilt krijgen, voeg je nog de opdracht Clrscr; toe.

Textcolor
Textcolor werkt net als textbackground. Een cijfer geeft aan welke kleur de letters worden.
Hierbij worden dezelfde cijfers gebruikt als bij textbackground.
Syntax:

```
textcolor(15);
```

Na dit korte regeltje zullen de letters wit zijn.

```
Program joehoe;
Uses crt;

Begin
Textbackground(5);
Textcolor(0);
ClrScr;
GotoXY(35,10);
Writeln('JOEHOEOEOEOEOE');
Readln;
End.
```

Opdr1_c

Schrijf zelf een programmaatje dat drie verschillende schermkleuren laat zien en de text mag je zelf verzinnen.
Zorg ervoor dat je een ClrScr; zet na de bepaling van de textbackground, dan wordt je hele scherm dezelfde kleur.

Lekker piepen in Turbo Pascal;
Sound
Met de opdracht Sound kan je de computer laten piepen via de systeemluidspreker.

Met behulp van Delay kan je een geluid even laten duren en met de opdracht nosound geef je de opdracht om weer te stoppen.
Syntax:

```
Sound(880);
```

Het getal dat tussen de haakjes staat geeft aan op hoeveel hertz er gepiept wordt. Je kan dus ook de toonhoogte van de piep bepalen.

Nosound
Nosound gebruik je als je wilt dat het gepiep dat je veroorzaakt hebt met sound wilt stoppen
Syntax:

Nosound;

Je gebruikt deze opdracht dus eigenlijk nooit los.

Bv

```
Program Pieper;
Uses crt;

Begin;
Clrscr;
Writeln('piep1');
Sound(880);
Delay(500);
Nosound;
Writeln('piep2');
Delay(500);
Nosound;
End.
```

Opdr1_d

Schrijf zelf een programma dat boer-er-ligt-een-kip-in-'twater-boer piept
En er de text bij laat zien in het midden van het scherm.
Je bent een bikkel als je er ook nog kleurtjes in verwerkt.
De toonhoogtes zijn(je hebt er 5 nodig)
Boer = 264 hertz (een c of do)
Er = 297 hertz (een d of re)
Ligt = 330 hertz (een e of mi)
Een = 352 hertz (een f of fa)
Kip = 395 hertz (een g of sol)
In = 352 hertz
'twa = 330 hertz
ter = 297 hertz
Boer = 264 hertz

Nog wat moois, de window.
window

Met de opdracht Window kan je je gebruiksveld in je programma wat verkleinen.
Syntax:

Window (1,1,5,5);

Deze opdracht zorgt voor een window in de linkerbovenhoek van je scherm. De eerste twee getallen zijn de x- en de y-coordinaten van de linkerbovenhoek van je window.
De tweede twee getallen (in dit voorbeeld de twee 5-en) zijn de x, en y-coordinaat van de rechteronder hoek.
Bv.

```
program windoos;
uses crt;

begin
textbackground(1);
clrscr;
window(10,4,30,13);
textbackground(4);
textcolor(15);
clrscr;
Writeln('een window');
window(30,13,40,20);
textbackground(6);
textcolor(15);
clrscr;
Writeln('nog een window waar wat in kan, maar gaat het er allemaal wel inpassen?');
readln;
end.
```

Opdr1_e
Maak een scherm met in het midden een window van 15 breed en 20 hoog. Zet in alle hoeken van de window een * (sterretje);

Kommentaar??

Kommentaar mag bij turbo pascal natuurlijk niet te lezen zijn voor de compiler, die snapt daar toch niets van. Mensen die elkaars programma willen uitbreiden of gewoon even doorlezen, willen graag soms een beetje uitleg over wat er waar gebeurt in een lang programma.

Als je graag even wat kommentaar kwijt wil dan kan dat tussen **{ }**(accolades).
De compiler slaat het gedeelte tussen de accolades gewoon over bij het compileren.

Hoofdstuk 2 VARIABELEN I

Turbo Pascal en eigenlijk alle andere programmeer-talen zijn natuurlijk bedacht door wis, en natuur-kundigen. Die gasten zijn natuurlijk dol op getallen en de leuke dingen die je daarmee kan doen. Er zal dus binnen het programeren heel veel met getallen moeten worden gewerkt.
Nu zijn we daar al een beetje aan gewend; kleuren hebben getallen, posities van het scherm bestaan uit getallen en geluid bestaat uit getallen.
Nu worden binnen een programma ook allerlei getallen gebruikt om dingen door te geven.
Het wordt ons als mens makkelijker gemaakt om met die getallen te werken door gewoon woorden en letters te gebruiken om daar een getal ‘in te stoppen’ en zo te kunnen onthouden. Zo weten we, binnen grote programma’s welke term nu wat betekent.
Een voorbeeld;

```
1     Program v_ari_abel;
2     Uses crt;
3     Var
4     NAAM:string;

5     begin
6     Clrscr;
7     writeln('Gimme een naam:');
8     readln(NAAM);
9     gotoXY(35,10);
10    writeln(NAAM,' is gek.');
11    readln;
12    end.
```

Nou, ik begrijp dat ik hier een heleboel te verklaren heb. Dus dat ga ik maar eens doen.
In regel 1 staat, zoals vanouds, de naam van het programma met ‘program’ er voor en een ; (puntkomma) erachter
Regel 2 deze kennen we ook al de Unit CRT wordt bij dit programma gebruikt.
Regel 3 Hier staat het woordje Var dat staat voor ‘variabelen’. Na Var geef ik een woord of letter op die ik binnen het programma wil gebruiken om iets ‘in te stoppen’.
Ik kan binnen een programma ook meerdere variabelen nodig hebben en hier declareren.
Regel 4: in dit geval noem ik de variabele ‘NAAM’. Daarachter type ik een : (dubbele punt) en dan het woordje ‘string’. Dit heeft niets met ondergoed te maken maar met de types variabelen.
String is een type die woorden en zelfs hele zinnen kan bevatten.
Er zijn verschillende types variabelen.
Maar daarover later meer.
Regel 5 geeft aan dat we hier met het echte programma beginnen.
Regel 6 zorgt er voor dat we weer met een net schoon scherm beginnen
Regel 7 geeft de schermkijker (jou) een opdracht om een naam in te typen
Hopelijk doe je dat ook want daar zit readln op te wachten.
In regel 8 zorgt readln er voor dat de ingetypte naam, in de variabele NAAM wordt gestopt.
Regel 9 zorgt voor de juiste cursorpositie.
En regel 10 is een write opdracht die dus de text op het scherm zet. Let op dat NAAM niet tussen aanhalingstekens staat omdat dat er voor zorgt dat het letterlijk wordt weergegeven.
Eigenlijk geven we write hier 2 opdrachten namelijk eerst moet hij de inhoud van de variabele NAAM op het scherm hoesten en daarna moet hij de letterlijke text weergeven. Deze twee opdrachten staan samen tussen haakjes maar zijn wel gescheiden door een komma.
Regel 11 is een gebruik van de ‘oude vertrouwde’ readln. Wachten tot er een toetsaanslaag komt.
Om daarna naar Regel 12 te kunnen.
Regel 12 The END.

Var
Var gebruik je in het begin van een programma of een procedure. Na var, geef je aan van welk type de variabelen zijn die je wilt gebruiken.
Syntax:

```
Var     regel_letters        : string;
        Getal1,getal2        : integer;
        Letter               : char;
```

Eerst geef je dus de naam van een variabele, dan een : (dubbele punt) en dan het type waartoe de variabele behoort. Als je meerdere variabelen hebt van een soort type kunnen deze achter elkaar getypt worden en gescheiden met een , (komma).

Readln II

Je hebt al te maken gehad met readln, toen gebruikte je die opdracht om het programma even te laten wachten tot er een toets ingedrukt werd zodat je programma niet afsloot voordat je kon zien wat er op het scherm gebeurt was.
Readln betekent eigenlijk: lees regel.
Readln kan lezen wat je op het scherm hebt getypt.
Deze vorm van readln wacht niet zomaar op een willekeurige toets.Op deze manier lees je vanaf het scherm wat een tieper heeft getyped.
Syntax:

```
Readln(bewaar);
```

Tussen haakjes staat een variabele, in ‘bewaar’ worden de gegevens bewaard die later in het programma weer nodig zijn.

Var

Bij Var ‘declareer’ je bepaalde variabelen. Je vertelt eigenlijk tegen de compiler dat er bepaalde woorden in de tekst zitten die hij niet kent. Bij bovenstaande voorbeeld heb ik verteld dat ik een variabele ga gebruiken van het
type String. In een string kunnen 255 lettertekens opgeslagen worden. Ze zullen letterlijk bewaard blijven totdat ze weer nodig zijn. Als de compiler maar weet wat voor soort variabele het is, dan vindt ie het goed als je het woordje NAAM ook in de programma-tekst gebruikt.
Je kan het woord NAAM ook veranderen in bijvoorbeeld SINTERKLAAS. Probeer maar.
Meerdere variabelen declareren gaat als volgt.

```
Var
Hond,kat,broer,zus:string;
```

Als ik dit boven een programma zet kan ik met behulp van readln de naam van de hond, de kat de broer en zus van een computeraar onthouden. En later met behulp van writeln weer op het scherm zetten bij voorbeeld:
Writeln('Je zus',zus,' lijkt op ',hond);

Opdr2_a
Schrijf een programma dat iemand om zijn naam, emailadres en zijn huisadres vraagt.
Laat daarna het scherm schoonvegen met clrscr; waarna op het scherm de volgende zin moet verschijnen.

Hallo NAAM, jouw huis woont op STRAAT, nummer NUMMER.
Ik ga de emailbox van MAILADRES volspammen.

Of iets soortgelijks.

Het **type integer**.
In de vorige opdracht heb je met behulp van een variabele met het type string informatie bewaard om die later weer te gebruiken.
De informatie die in een string staat zijn eigenlijk alleen maar tekens die weer op het scherm kunnen verschijnen.
Programmeren is iets voor cijfer boys en girls dus zijn er vooral veel types bedacht die cijfers kunnen onthouden.
Dit is bijvoorbeeld het geval met het type integer. Een variabele GETAL van het type integer kan getallen bewaren. In een variabele van het type integer kan een geheel getal (dus niets na de komma) **tussen de 32767 en de min 32767** (-32767).
Waarom dit zo is ga ik niet proberen uit te leggen omdat ik dat veel te ingewikkeld vind en je hoeft niet zo nodig te weten waarom het zo is als je maar nooit probeert een getal dat groter is dan 32767 bijvoorbeeld 2 miljoen door een integer te laten bewaren.

```
1     Program rekenen;
2     Uses CRT;
3     Var UITKOMST,GETAL1,GETAL2:integer;
4         NAAM                :string;
5     Begin
6     Clrscr;
7     Writeln('Hallo, hoe heet jij?');
8     Readln(NAAM);
9     Writeln('Gimme een getal');
10    Readln(GETAL1);
11    Writeln('Doe dat nog eens');
12    Readln(GETAL2);
13    UITKOMST:=GETAL1+GETAL2;
14    Writeln('Hoi ',NAAM,' de som der cijfers is : ',UITKOMST);
```

```
15      ReadIn;
16      End.
```

Ik declareer dus drie integers en een string voor dit programmaatje voordat ik begin met BEGIN
In regel 13 stop ik de uitkomst, in de variabele UITKOMST, dit met behulp van de dubbele punt en = (het is-teken).

Het verschil tussen = en :=
Het is-gelijk-teken (**=**) dient om dingen te vergelijken dus in een vraagstelling (zie if then else).
Bv.

```
If x=0 then writeln('das niet veel');
```

Het is-gelijk-teken met de dubbele punt ervoor dient niet om te vragen maar om mee te delen. Je zegt tegen de computer dat het zo is.
Bv. **piek:=100;**
Dit is een mededeling je moet het eigenlijk van achter naar voren lezen; het getal 100 wordt in de variabele piek gezet.

Op de lagere school hebben we geleerd dat twee getallen met het + (plus-teken), bij elkaar worden opgeteld. We hebben nog wat meer geleerd in die reken lessen en dat kunnen we nu leuk gebruiken in ons turbo pascal programmaatje.

/	betekend gedeeld door.
*****	is maal
-	is min

Opdr2_b
Maak nu zelf een programma dat van twee ingevoerde getallen, de som, het product, het verschil en de breuk uitrekend. Denk eraan bij het invullen van de twee getallen dat de gedeclareerde integer geen breuken aankan!

Real
Natuurlijk kan er wel met breuken of te wel 'gebroken getallen' gewerkt worden.
Bijvoorbeeld met een variabele van het **type Real**
Een variabele van het type real kan elf cijfers nauwkeurig bevatten. Het maakt niet uit waar de komma (in Turbo Pascal een punt!!) gezet wordt. Een real snapt het allemaal.
Probeer het volgende maar eens uit.

```
Program met_een_punt;
```

```
Uses crt;
Var getal:real;

Begin
Getal:=12345.6789012;
Writeln(Getal);
Getal:=12.12345678;
Writeln(Getal);
Getal:=0.00123456789
Writeln(Getal);
Readln;
End.
```

Zoals je ziet krijg je niet echt een lekkere notatie van de getallen op je scherm. Het programma maakt er allemaal wetenschappelijke notatie-dingen van.
Om het allemaal wat netter te maken heeft turbo Pascal de mogelijkheid om aan te geven hoeveel cijfers er voor en hoeveel er na de komma (punt) komen.
Verander bovenstaande eens in:

```
Program met_een_punt;
Uses crt;
Var getal:real;

Begin
Getal:=12345.6789012;
Writeln(Getal:5:2);
Getal:=12.12345678;
Writeln(Getal:2:8);
Getal:=0.00123456789
Writeln(Getal:1:10);
Readln;
End.
```

Zo kan je de dingen dus wat netter maken. Het eerste getal geeft aan hoeveel cijfers er voor de komma mogen staan en het tweed cijfer geeft aan hoeveel er na de komma mogen staan.
In totaal kunnen er maar 11 cijfers in een real dus houd daar rekening mee!!

Opdr2_C
Verander Opdr3_b eens zo dat je met goed fatsoen ook uitkomsten met 3 getallen achter de komma kan krijgen.

Hoofdstuk 3 IF then ELSE

We hebben al een paar interactieve programma's gemaakt maar dit kan nog veel leuker.
We kunnen een programma om aktie van de gebruiker laten vragen en aan de hand van die aktie bepalen wat we verder gaan doen.
Type onderstaande over en run het.

```
1       program als_jij_dit_doet_dan_doe_ik_dat;
2       uses crt;
3       var
4       getal:integer;

5       begin
6       clrscr;
7       writeln('Gimme een geheel getal onder de tien: ');
8       readln(getal);
9       if getal<10 then
10      writeln('Geweldig je bent een genie!')
11              else
12              writeln('Vraag schoolgeld van afgelopen jaren terug');
13              sound(100);
14              delay(200);
15              nosound;
16              readln;
17      end.
```

In Regel 4 'declareren' we de variabele 'getal'.
In Regel 8 stoppen we hetgeen de computeraar intypt, in de variabele 'getal'
Regel 9 bevat de if then vergelijking. Dit bestaat uit het woordje **IF** dan komt er een vergelijking, in dit geval vraag ik of het ingevoerde getal kleiner is dan 10. Als dat zo is moet het programma doen wat na then komt. Hij gaat dus op het scherm schrijven 'Geweldig je bent een genie!'.
Als het niet klopt wat er na if komt dus als je het getal 12 hebt ingevuld, dan gaat het programma automatisch naar else en gaat doen wat er na else komt.
Let op voor else komt nooit een ; (puntkomma)!
Na then ook niet!

If then else
'If …then…. Else…'betekend in gewoon Nederlands gewoon 'als…dan …en anders…'
Na if komt eerst een voorwaarde, dus bijvoorbeeld
if annie=tante **then** nodig uit op mijn verjaardag.

Else
Nodig niet uit
Syntax:
If x=2 then
 writeln(‘het is twee!!’)
else
writeln(‘het is geen twee!!’);
Je ziet dat na het woordje then, geen ;(puntkomma) komt.
Voor else komt ook geen ;(puntkomma) na else ook geen ;(puntkomma).

Opg3_a
Herschrijf bovenstaande programma zo dat er naar een wachtwoord (string) gevraagd wordt.
Schrijf na BEGIN op de eerstvolgende regels:

```
clrscr;
wachtwoord:='koetjekoe';
```

daarmee stel je een wachtwoord in op koetjekoe.
Geef een positieve melding als het goed gaat en een negatieve mededeling als het fout gaat.

Nog meer if then else: I
Soms wil je dat je programma meerdere regels uitvoert na een if en ook meerdere na een else.
Dan moet je gebruik gaan maken van **begin en end maar dan op een andere manier**. Zie het volgende voorbeeld.

```
1       program met_meerdere_beginnnen_en_einden;
2       uses crt;
3       var
4       getal:integer;

5       begin
6       clrscr;
7       writeln('Gimme een geheel getal onder de tien: ');
8       readln(getal);
9               if getal<10 then
10              begin
11              textbackground(5);
12              textcolor(0);clrscr;
13              gotoXY(35,10);
14              writeln('Geweldig je bent een genie!');
15              sound(880);
16              delay(500);
17              nosound;
18              clrscr;
19              end
20                      else
21                      begin
22                      textbackground(4);
23                      textcolor(14);clrscr;
24                      writeln('Vraag schoolgeld van afgelopen jaren
terug');
25                      sound(100);
26                      delay(500);
27                      nosound;
28                      readln;
29                      end;
30      end.
```

Je ziet dat er een soort sub-programmaatjes zijn gemaakt die beginnen allebei met ‘begin’ en eindigen met end;.
Het is belangrijk dat deze ‘tussen end;’s’ eindigen op een ; (puntkomma) en dus niet op een . (punt). De end met de punt betekend echt het einde van het

programma. De end op regel 19 heeft geen ; (puntkomma), dat komt door else voor else komt nooit een ; (puntkomma).
Op regel 12 en 23 is ook iets vreemds aan de hand. Je ziet dat ik daar meerdere opdrachten op een regel heb getypt. Dat mag, als je ze maar scheidt met ;; (puntkomma's).
Voor alle duidelijkheid:
Met een if then else vergelijking kijk je of er aan bepaalde voorwaarden voldaan wordt. Op grond daarvan laat je het programma uitvoeren wat achter if then staat, of wat achter else staat.
Als die opdrachten meerdere regels bevatten zet je die regels tussen 'begin' en 'end'.
Denk aan ;; (puntkomma's) en waar ze wel en niet moeten. Het geeft niet zo heel erg als je er een vergeet. De compiler geeft vanzelf een foutmelding.

Opg 3
Schrijf een programma dat vraagt om een achtergrondkleur en een textkleur. Zorg dat de rest van je programma die kleur heeft en laat een gebruiker kiezen tussen een goede voetbalploeg en een slechte. Als de gebruiker(computergebruiker) een goede ploeg kiest moet je computer feestelijk juichen (piepjes en leuke kleuren) anders moet het donker en met diepe tonen antwoorden.

Nog meer if then else II
Je kan natuurlijk ook meerder if else vergelijkingen gebruiken of meerdere ifs en een else.
Let op je kan else ook helemaal weglaten!

```
Program wat_een_boel_ifjes;
Uses crt;
Var getal:integer;
Begin
Writeln('kies een getal onder de drie en boven de een :');
Readln(getal);
If getal=2 then
Writeln('in een keer goed');
If getal<2 then
Writeln('nee veeeeel te laag')
If getal>2 then
Writeln('Nee veeeeel te hoog');
Readln;
End.
```

Je ziet dat er geen else bijstaat. Dat hoeft ook niet maar is wel handig om alle andere mogelijkheden op te vangen. Een else hoort altijd bij de laatste If!!! Run het volgende programma om te zien wat ik hiermee bedoel. Probeer het programma uit met 1, 2, 3 en 10!

```
Program wat_een_boel_if_en_else;
Uses crt;
Var getal:integer;
Begin
Writeln('kies een getal onder de vier en boven de nul :');
Readln(getal);
If getal=2 then
Writeln('in een keer goed');
If getal<2 then
Writeln('nee veeeeel te laag');
If getal>2 then
```

```
Writeln('Nee veeeeel te hoog')
Else
Writeln('je bent een oen');
Readln;
End.
```

Je ziet dat ik kan vragen of getal gelijk is aan twee maar ook het > (groter-dan-teken) en het < (kleiner-dan-teken) zijn te gebruiken om te zien aan welke voorwaarden getal voldoet.
=, <, > noem je vergelijkings operanden. Je vergelijkt twee dingen met elkaar.

Het volgende programma laat zien wat een puinhoop het kan worden met if en else.
Als je dat kan overtiepen mag je dat doen.
Ik raad het persoonlijk niet aan maar het mag als je een echte logica-super-hero wilt worden

```
Program wat_een_zooi_met_if_en_else;
Uses crt;
Var getal:integer;
Begin
Clrscr;
Writeln('Kies een getal onder de vier en boven de nul :');
Readln(getal);
If getal=2 then
Writeln('in een keer goed');
If getal<2 then        begin
                Writeln('nee veeeeel te laag');
                Writeln('probeer nog eens');
                Readln(getal);
                If getal =2 then
                Writeln('nou nou in twee keer goed');
                If getal>2 then        begin
                                Writeln('ik geef je nog  een kans');
                                Writeln('geef me nog een getal');
                                Readln(getal);
                                If getal=2 then
                                Writeln('je bent geniaal')
                                Else
                                Writeln('this is not your lucky day');
                                End;
                End;
If getal>2 then begin
                        Writeln('Nee veeeeel te hoog');
                        Writeln('probeer dat nog eens');
                        Readln(getal);
                        If getal =2 then
                        Writeln('Nou nou in twee  keer goed dat is
fantastisch');
                        If getal<2 then     begin
                                        Writeln('Ik geef je nog  een
kansje.');
                                        Writeln('Geef me nog een getal.');
                                        Readln(getal);
                                        If getal=2 then
                                        Writeln('well done Einstein')
                                        Else
                                        Writeln('do not go to a casino
today');
```

```
                End;

            end;
ReadIn;
End.
```

Gelukkig hebben de makers van Turbo Pascal iets gevonden om dit soort letterdoolhoven te vermijden.

Het **While do** commando

```
1       Program terwijl_het_zo_is_doe_je_dat;
2       Uses crt;
3       Var getal:integer;
4       Begin
5       Getal:=100;
6       Clrscr;
7       While getal<>2 do
8               begin
9               Writeln('Gimme een getal');
10              Readln(getal);
11              If getal=2 then
12              Writeln('allllrighty then')
13              Else
14              Writeln('doe da nog es');
15              End;
16      Readln;
17      End.
```

Je ziet dat ik eerst in regel 5 de variabele 'getal' de waarde 100 heb gegeven. Die waarde verandert na de eerste keer dat iemand een getal heeft opgegeven. In regel 7 zeg ik eigenlijk: Terwijl de variabele 'getal' groter of kleiner is dan 2, voer de opdrachten tussen begin en end uit.

Opg4_a

Maak nu zelf een programma dat in oneindigheid om een password blijft vragen totdat het juiste wordt ingevoerd. Let op dat je dat password declareert als string!!

Opg4_b

Maak een programma waar je if then else in combineerd met een while do lus. Vraag of iemand een getal wil raden of een wachtwoord en zorg er voor dat er dan ook een gepaste reactie komt.

```
Program maar_drie_pogingen;
Uses crt;
Var getal,teller:integer;
Begin
Getal:=100;
teller:=1;
Clrscr;
Writeln('Gimme een getal,dit is poging:',teller);
While (getal<>2) and (teller<>4) do
    begin
    Readln(getal);
    teller:=teller+1;
                If getal=2 then
```

```
                Writeln('allllrighty then, het is je gelukt in ',teller-
1,'pogingen')
                Else
                Writeln('Do da nog es je hebt nog ',4-teller,' pogingen');
    end;
    readln;
End.
```

Typ dit programma over en kijk wat er gebeurd.
Je ziet dat er bij while nu twee voorwaarden zijn daartussen staat het woordje and. ‘And’ betekend ‘en’ in het nederlands. Wat er hier dus gezegd wordt is het volgende:
Terwijl het getal geen 2 is **en** teller is nog geen 4, doe dat leuke raadspelletje.
Behalve dat er twee voorwaarden zijn is er nog iets bijgekomen, namelijk het gebruik van een teller. Teller is een variabele die elke keer dat de lus doorlopen word, met 1 in waarde stijgt.
Turbo Pascal ziet dat woordje teller, dat als integer is gedeclareerd, als een getal. Dus kan je het gewoon ook in berekeningen gebruiken. Bijvoorbeeld letterlijk als teller-1 is de waarde van teller **op dat moment** verminderd met 1.

```
Program de_tafel_van;
Uses crt;
Var teller,getal:integer;

Begin
clrscr
Teller:=1;
Getal:=4;
While teller<11 do
Begin
Writeln(getal*teller);
Teller:=teller+1;
End;
Readln;
End.
```

Opg4_c
Verander het bovenstaande programmaatje zo dat het er een beetje fatsoenlijk uitziet dus:
Eerst een zinnetje met: De tafel van 4.
1 keer 4 is: 4
2 keer 4 is: 8 etc..

Opg4_d
Maak een programma dat eerst om een getal vraagt en dan daarvan de tafel op het scherm hoest.

Opg4_e
Maak een programma dat eerst vraagt om een text en daarna om een getal.

Met deze twee gegevens laat het programma een aantal(getal) keer de text(text) op het scherm verschijnen.

Case of

Bij if /else en bij while do vragen we om voorwaarden.
Je kan je voorstellen dat er soms zo veel voorwaarden zijn dat het niet meer lekker loopt met al die while-lussen en if else vergelijkingen.
Om eens even wat leuks te doen en gelijk de case of functie te leren gaan we een klein pianootje maken van het toetsenbord. Schrik niet we gaan ook even een beginnetje maken met het maken van een procedure.
Type eerst maar over en zie wat er gebeurt.

```
1       {Met dit programma gaan we muziek maken}
2       Program muziek;
3       uses Crt;

4       var
5        C: Char;


6       procedure piep(toon:word);
7       begin
8       sound(toon);
9       delay(200);
10      nosound;
11      end;

12      begin
13           Textbackground(4);
14           TextColor(15);
15           ClrScr;
16
GotoXY(20,5);write('*****************************************
*');
17           GotoXY(20,6);write('* Je keyboard is nu een synthesizer  *');
18           GotoXY(20,7);write('*van de z linksonder op je keyboard  *');
19           GotoXY(20,8);write('* tot en met de komma rechts onder  *');
20           GotoXY(20,9);write('* op je keyboard, piepen de toetsen   *');
21           GotoXY(20,10);write('*                                    *');
22           GotoXY(20,11);write('* midden onder in het scherm          *');
23           GotoXY(20,12);write('* zie je welke toets je indrukt
*');
24           GotoXY(20,13);write('* Stoppen doe je met  de Q!!!         *');
25           GotoXY(20,14);write('*   Veel plezier!!!!                  *');
26
GotoXY(20,15);write('*****************************************
***');

27       while Ord(C)<>113 do {113 is de q }
```

```
28          begin
29          GotoXY(35,20);Writeln( C);
30          C:= Readkey;
31          case  Ord(C) of
32          44  :piep(600);  {,,c}
33          109 :piep(500);  {m,b}
34          110 :piep(440);  {n,a}
35          98  :piep(400);  {b,g}
36          118 :piep(359);  {v,f}
37          99  :piep(300);  {c,e}
38          120 :piep(250);  {x,d}
39          122 :piep(200);  {z,c}
40          end;
41      end;
42      end.
```

In **regel 1** zie je dat er commentaar staat tussen twee { }(accolades). Dat mag dus in Turbo Pascal. Je kan hele verhalen vertellen en de compiler stoort zich daar niet aan als het geheel maar
tussen { } (accolades) staat. Deze mogelijkheid kan je gebruiken om de code van lange onoverzichtelijke programma's een beetje duidelijk te houden voor diegene die er misschien wat in wil veranderen.

goede programmeurs
zijn lui en maken dus veel gebruik van procedures en functies

De **regels 6 t/m 11** bevatten een procedure. Een procedure is een soort programma in een programma. Zoals je ziet is de eerste regel weer een naam met een ; (puntkomma) erachter.
We zien alleen dat er voor die ; (puntkomma) nog een variabele wordt gedeclareerd tussen twee ( ) haakjes. In dit geval heet de procedure: 'piep'
de variabele : 'toon' die is gedeclareerd als word
Word is een type dat een geheel getal kan bevatten tussen de 0 en de 65536
De variabele 'toon' is alleen maar nodig in de procedure en daarom gaan we hem ook niet bovenaan in het hoofdprogramma declareren.
Een procedure heeft weer netjes een begin (**regel 7**) en een eind (**regel 11**). Daarbinnen kan weer alles gebeuren wat in een gewoon programma ook gebeurt.
De Ord functie haalt de getalswaarde uit een opgegeven variabele. In dit geval de getalswaarde van een toets van je keyboard. De functie Readkey gaat zitten wachten tot je een toets indrukt. Als dat bijvoorbeeld een 'z' is dan stopt ie dat in de variabele C. Ord(C) haalt dus de getalswaarde uit 'z'. Alle tekens hebben een getalswaarde, zie de asccii tabel die je hieronder kan overtypen.
Regel 32 t/m 39 geven precies een toonladder weer die van c t/m c loopt. Het zeer korte commentaar achter de regeltjes vertellen welke toets het op je keyboard is en welke toon uit de toonladder het moet worden.
De voorwaarde van de while lus is dat de aangeslagen toets geen 'q' mag zijn, dan stopt het programma. Dat is natuurlijk wel handig, om ook te kunnen stoppen.
Misschien is je ook opgevallen dat hier niet de functie 'writeln' is gebruikt maar de functie 'write'. Ze doen eigenlijk allebei hetzelfde alleen write geeft niet automatisch een nieuwe regel voor de volgende write opdracht. ln staat dus voor line dat engels is voor regel.

Procedure

Procedures worden dus gebruikt om veel gebruikte taken uit je programma maar een keer te hoeven beschrijven.
Syntax:

```
Procedure doe_eens_wat(getal1,getal2:integer);
Var getal3:integer;
Begin
Getal3:=getal1+getal2;
Writeln('optellen geeft :',getal3);
Readln;
End;
```

De eerste regel bestaat uit het woord procedure gevolgt door de naam. Achter de naam staan de twee variabelen die 'gevuld' moeten worden vanuit het hoofdprogramma met bijvoorbeeld de code:
Doe_eens_wat(8,90);
Net als in een hoofdprogramma kan je hier ook variabelen declareren met het woordje var.
Deze variabele telt alleen binnen deze procedure!! Hij is lokaal gedeclareerd !!

Functie
Een functie kan je, net als een procedure gebruiken om vaker voorkomende taken maar een keer uit te hoeven schrijven.
Syntax:

```
Function doe_eens_wat(getal1,getal2:integer)integer;
Begin
Doe_eens_wat:=getal1+getal2;
End;
```

Een functie ziet er eigenlijk hetzelfde uit als een procedure, het gebruik is echter anders.
Een procedure doet zelf iets met die gegevens en die gegevens blijven lokaal, in de procedure aanwezig. Gebruik je een functie, dan is het de bedoeling dat je de uitkomst van de functie doorgeeft aan het hoofprogramma.
In het hoofdprogramma zou je dus de volgende zin kunnen verwachten.

```
Writeln('optellen geeft : ',doe_eens_wat(8,90));
```

```
program de_ascii_tabel;
uses crt;
var x,y, getal:integer;

procedure letter_en_getal( g:integer);
begin
gotoXY(x,y);
write(' ',g);
write('=',Chr(g));
inc(y)
end;
procedure rij(een,twee:integer);
begin
for getal:= een to twee do
letter_en_getal(getal);
x:=x+7;
y:=1
end;
begin
clrscr;
x:=1;y:=1;
rij(1,23); rij(24,46); rij(47,70); rij(71,94); rij(95,118); rij(119,142);
rij(143,166); rij(167,190); rij(191,214); rij(215,238); rij(239,255);
readln
end.
```

Typ het programma ‘de ascii tabel’ en zie hoe er gewerkt wordt met twee procedures.

Dat er geen ; (puntkomma) staat achter de laatste opdracht voor een end; of end. is geen schoonheidsfoutje maar is eigenlijk veel netter. Het zorgt er voor dat de compiler niet meer naar opdrachten zoekt.

```
program wat_zei_je;
uses crt;
var iets:string;

procedure kuch(wat:string);
begin
writeln(‘ik typte : ’,wat);
end;

begin
clrscr;
writeln(‘Typ eens wat…..:’);
readln(iets);
kuch(iets);
readln
end.
```

Je ziet dat er in het hoofdprogramma wat word opgevraagd. Het ‘aanroepen van de procedure ‘kuch’ met de variabele iets gebeurt. Als de aanroep ‘kuch’ boven aankomt en hij neemt ‘iets’ mee dan wordt dat ‘iets’ automatisch in de variabele ‘wat’ gestopt. Daarmee gaat de procedure dan aan de gang.

Opg5_a

Maak een programma dat vraagt om een getal.
In een procedure moet je 5 bij dat getal optellen en dit op het scherm laten zien.

Opg5_b

Maak een programma dat vraagt om getallen die de kleuren voorstellen.
Geef keuze uit 5 kleuren en laat het scherm die kleur worden met text in het midden.
bijvoorbeeld:

```
geef een getal dat een kleur betekent: 4
lees kleur;                            rood
```

nu moet er een rood scherm verschijnen met de zwarte of witte tekst ‘rood’ in het midden.

Repeat

De functie Repeat kan ook gebruikt worden om een handeling een aantal keer achter elkaar te laten herhalen totdat er aan een voorwaarde is voldaan.

```
Program nog_eens;
uses crt;
var keuze:integer;
begin
clrscr;
writeln('Geef eens een getal onder de 100');
readln(keuze);
repeat
writeln('Nog een, het is niet die ik bedoel')
readln(keuze);
until keuze=20;
writeln('Jaaaaaa die is goed');
readln
end.
```

Opg5_a
Herschrijf dit programmaatje eens zodat er om een woord gevraagd wordt totdat het goede woord is ingetypt.

Opg5_b
Herschrijf dat programmaatje met behulp van een teller zodat er maar 5 keer gegokt mag worden.
Het is niet erg om nog een paar keer te mogen gokken als je het goed hebt voordat die 5 keer om zijn!

```
program casino;
uses crt;
var gok,getal:integer;

begin
clrscr;
randomize;
getal:=random(100)
writeln('raad eens welk getal ik in mijn schijf heb..:');
readln(gok);
if gok=getal then
writeln('je bent goed')
else
writeln('no way');
readln;
```

```
end.
```

Zie hier de functie Randomize; en het gebruik van random.
Dit programaatje hoest een willekeurig geheel getal op onder de honderd
en laat iemand er naar raden.
Probeer dit eens zo uit te werken zodat er een programma ontstaat dat iemand
de kans geeft om vaker te gokken en bijhoud hoeveel keer er gegokt is.

Nog eens procedures en functies.

Een procedure is een soort ‘onderprogrammaatje’ dat een vaker voorkomende taak kan uitvoeren. Deze taak hoef je dan dus niet elke keer opnieuw in te typen en je programma kan er overzichtelijker van worden.
Een functie is net als een procedure een soort ‘onderprogramma’, het verschil met een procedure is dat een functie een waarde moet teruggeven aan het programma. Die waarde moet natuurlijk van een bepaald type zijn (integer, real, word)dat moet je ook aangeven in de eerste regel van de functie.
Zie het programma hieronder voor de verschillen tussen functies en procedures.

```
Program mijn_eerste_rekenmachine;
Uses crt;
Var a,b,uitkomst:real;
    Keuze:integer;
Procedure ga_schrijf(x,y:integer;tekst:string);
begin
GotoXY(x,y);write(tekst);
End;

Procedure keuzelijst;
Begin
Ga_schrijf(10,10,'Wilt u optellen kies 1');
Ga_schrijf(10,12,'Wilt u aftrekken kies 2');
End;

Function optellen(a,b:real):real;
begin
Uitkomst:=a+b;
End;

Function aftrekken(a,b:real):real;
Begin
Uitkomst:=a-b;
End;


Begin
        clrscr;
        Ga_schrijf(10,6,'gimme een getal');
        Readln(a);
        Ga_schrijf(10,8,'Mag ik er nog een');
        Readln(b);
        Keuzelij        st;
        Readln(keuze);
```

```
case keuze of
1 : optellen(a,b);
2 : aftrekken(a,b);
        else
        Writeln('Dat getal ken ik niet..');
        End;
        Writeln('De uitkomst is : ', uitkomst:6:6);
readln
End.
```

Opg5_b
Schrijf dit programma over en zorg er voor dat er nog twee functies bijkomen die kunnen vermenigvuldigen (*) en delen (/).
Pas natuurlijk ook het keuze menu aan en het ‘case of-gedeelte’.

Opg5_c
Herschrijf Opg1_d zo dat je het piepen in een procedure zet en aanroept met bv.
Piep(1000);
Piep(1200);
Etc…

```
Program newsflash;
Uses crt;
Begin
Textbackground(4);textcolor(15);clrscr;
GotoXY(20,10);writeln('    ER IS EEN ERG FATALE FOUT OPGETREDEN
');
GotoXY(20,12);writeln('UW COMPUTER LEGT HET LOODJE BINNEN 5
SECONDEN');
Delay(3000);
Textbackground(0);clrscr;
Readln;
End.
```

Opg5_d
Maak een programma die het programma newsflash tot in het niets laat verbleken.
Maak procedures voor beeldscherm-verversing en geluid.
Het scherm moet beurtelings vijf keer oplichten met de mededeling dat er nog maar 4, 3, 2, 1, seconden over zijn. Maak hierbij dus gebruik van een teller.
Bij elk nieuw scherm moet ook een waarschuwingspiep te horen zijn.

Bij het zwarte scherm met niets erop moet een lange piep klinken die bijna niet ophoud (totdat je op een toets duwt natuurlijk. Stel zelf het delay zo af dat het precies 5 seconden in totaal is.

Array

Een variabele wordt altijd gedeclareerd, bij Var weet je wel?.., als een bepaald type.
De types die we tot nog toe gehad hebben zijn; Real, integer, word, string en char. De laatste twee uit dit rijtje zijn speciaal voor stukken text en letters.
De array is ook een type.
Een array bevat altijd een rij, die weer van een bepaald type zijn.
Een rij van –3 tot en met 5 van het type integer bevat de getallen –3, -2, -1, 0, 1, 2, 3, 4 en 5.
Een array is dus niet één getal maar een rij van een bepaald type.
Je kan in Turbo Pascal dus zelf bepalen welke getallen dat zijn.
Het eerste array waar we gebruik van zullen maken is een een-dimensionaal array.
Dit betekend dat we die array voor kunnen stellen als een rij met vakjes waar we dan waarden in kunnen stoppen.
Onze array van _3 tot en met 5 zal er dan dus zo uit zien.

-3	-2	-1	0	1	2	3	4	5

Als je dit dus declareerd krijg je dus:

Var onze_rij: array[-3..5] of integer;

Onze_rij houd dus die hele balk in, die hele balk met getallen krijgt maar één naam namelijk onze_rij.
Array[-3..5] betekend dat in onze_rij de getallen –3 tm 5 zitten.
Of integer houdt in dat inhoud van die vakjes getallen van het type integer zijn.

```
1    program een_array_is_een_rij;
2    uses crt;
3    var
4    teller:integer;
5    RIJ:array[1..9] of integer;
```

```
6       begin
7           clrscr;
8           rij[1]:=15;
9           rij[2]:=25;
10          rij[3]:=35;
11          rij[4]:=45;
12          rij[5]:=55;
13          rij[6]:=65;
14          rij[7]:=75;
15          rij[8]:=85;
16          rij[9]:=95;
17          begin

18              for teller:=1 to 9 do
19             write(RIJ[teller],' ');
20              teller:=teller+1
21              end;
22      readln
23      end.
```

Dit programma is een voorbeeldprogrammaatje van wat een je met zo'n array kan doen.
In regel 4 geef ik aan dat er een variabele is die teller heet. Deze is van het type integer. Een integer mag de waarde –32768 t/m 32767 hebben en is een geheel getal.

In regel 5 declareer ik een variabele rij, die van het type array is. Dan geef ik aan dat die array bestaat uit de getallen van 1 t/m 9. Eigenlijk worden er dus 9 vakjes gereserveerd, die later wel met iets gevuld zullen worden. De blokhaken waartussen ik aangeef dat de getallen van 1 t/m 9 mogen meedoen, zie je ook terug in regel 8 t/m 16 en 19. Omdat een array dus uit meerdere 'vakjes' bestaat, en je wilt een van die vakjes uitlezen, is het dus niet genoeg om te zeggen Writeln(rij). Het programma weet niet welk vakje van die rij je nu eigenlijk bedoeld dus er gebeurt niets.

In vakje rij[1] stop ik dus het getal 15.
De teller wordt later gebruikt om alle vakjes van de array uit te lezen.
Run het volgende programma eens en kijk wat er gebeurt.
Wat er op het scherm te zien is zijn natuurlijk de getallen die er in de plekken van de arree (10,20..etc) staan. Aangezien er ook nog vakjes tussen zitten zijn die leeg. Dit veroorzaakt die 00(nullen) op je scherm.
Hij wil namelijk ook de waarde van bijvoorbeeld vakje 11 op het scherm tonen, en dat vakje hebben we niet gevuld!

```
1       program een_array_is_een_rij;
2       uses crt;
3       var
4       teller:integer;
5       RIJ:array[10..90] of integer;

6       begin
7          clrscr;
8          rij[10]:=15;
9          rij[20]:=25;
10         rij[30]:=35;
11         rij[40]:=45;
12         rij[50]:=55;
13         rij[60]:=65;
14         rij[70]:=75;
15         rij[80]:=85;
16         rij[90]:=95;
17         begin

18          for teller:=10 to 90 do
19         write(RIJ[teller],' ');
20         teller:=teller+1
21         end;
22         readln
23         end.
```

Nu gaat het moeilijk worden let op let op!

Nu weten we dat het mogelijk is om vakjes te declareren. Het zijn dan wel virtuele vakjes maar die kan je vullen met informatie.
In de vorige oefening heb je gezien dat je bijvoorbeeld waarden van het type integer in zo'n vakje kan stoppen.
Ja leuk maar om nu een heel vak te reserveren voor alleen een getalletje. Dan kan je net zo goed het getal zelf typen.

Het is ook mogelijk om zelf een type te bedenken en die te maken. Zo vind ik het leuk om een type te bedenken dat ‘emailadres’ heet en dat de naam, nickname en emailadres van iemand kan bevatten.
Als ik dan bedenk dat ik die in vakjes kan bewaren dan heb ik dus zelf een soort adressenbestand geprogrammeerd.

Er heeft ook ooit iemand het type integer bedacht en ook het type string is ooit bedacht door een gewone simpele programmeur die wat leuks wilde doen met letters.
In turbo Pascal kan iedereen zomaar een leuk type verzinnen en dat gaan gebruiken in zijn of haar programma.
Het enige wat je moet doen is je aan de regels houden en die zijn niet zo heel erg moeilijk.

Bovenaan in het programma, nog voor het var-gedeelte waar je de variabelen declareerd, zet je zoiets als dit:

```
1       Type
2       Emailadres = record
3       Adres:string[30];
4       Naam:string[20]
5       Nickname:string[15]
6       Nummer: integer
7       End;
```

In regel 1 geef ik aan dat we hier een nieuw type gaan maken.
Regel 2 geeft aan dat dat type ‘emailadres’ gaat heten. =record betekend dat we dat nu gaan vullen. Dat kan je vergelijken met de ‘opneemtoets’ van een tape-recorder.
Vervolges ga ik in regel 3 t/m 6 aangeven welke dingen er allemaal in dat type zitten. Een emailadres dat ik adres noem de naam en de nickname van de persoon. In regel 6 zeg ik dat er ook bij iedereen een nummer gaat horen zodat ik later op nummer zou kunnen gaan zoeken.
Regel 7 bevat een end; dat betekend dat het opnemen met record nu is gestopt en dat nu het type af is. Er is een nieuw type gemaakt.

```
1       program gebruik_een_arree_en_record;

2       uses crt;

3       type
4       emailadressen=record
5       naam:string;
6       adres:string;
7       nickname:string;
8       nummer:integer;
9       end;

10      var emailadres:emailadressen;

11      begin
12      clrscr;
13      emailadres.naam:='piet';
14      emailadres.adres:='piet@honda.nl';
15      emailadres.nickname:='dikke piet';
16      emailadres.nummer:=1;

17      with emailadres do
18      begin
19      writeln('De naam is      : ',naam);
20      writeln('Het emailadres is: ',adres);
21      writeln('Nickname is     : ',nickname);
22      writeln('Het nummer is   : ',nummer);
23      end;
24      readln;
25      end.
```

Hierboven zie je een programma dat gebruik maakt van de opdracht record om een type te maken.
In regel 17 staat ‘with emailadres do’. Het type emailadres bestaat uit meerdere ‘velden’, er zit ten slotten een naam in, een adres, een nickname en een nummer. Bij with emailadres do leest ie dus alle velden uit die in dat type gevuld zijn in een eerder stukje programma (regel 13 t/m 16).

Opdr6_a
Voeg nog een eigenschap aan het type emailadres toe, bijvoorbeeld het item spreuk dat staat voor lijfspreuk van die persoon.
Emailadres.spreuk is natuurlijk een string.

```
1       program gebruik_een_arree_en_record;

2       uses crt;

3       type
4       emailadressen=record
5       naam:string;
6       adres:string;
7       nickname:string;
8       nummer:integer;
9       end;

10      adressenlijst=array [1..3] of emailadressen;

11      var
12          nr,welk :integer;
13          lijst      :adressenlijst;

14      begin

15      nr:=1;
16      for nr:=1 to 3 do
17      begin
18      clrscr;
19      writeln('naam        : ');
20      readln(lijst[nr].naam);
21      writeln('emailadres is: ');
22      readln(lijst[nr].adres);
24            writeln('nickname is  : ');
25            readln(lijst[nr].nickname);
25      lijst[nr].nummer:=nr;
26      end;

27      clrscr;
28      writeln('welk nummer wil je zien, je mag kiezen uit 1 t/m 3 ??');
29      readln(welk);
30      writeln('Naam is        : ',lijst[welk].naam);
31      writeln('Emailadres is   : ',lijst[welk].adres);
32      writeln('Nickname is     : ',lijst[welk].nickname);
33      writeln('en het nummer is: ',lijst[welk].nummer);
34      readln;
35      end.
```

In regel 3 t/m 9 maken we een nieuw type; emailadressen.
In regel 10 wordt een array aangegeven van 1 t/m 3. Dus er zijn drie vakjes in dat array. In elke van die vakjes kan een variabele van het type emailadressen

staan. Zo zie je dus dat er in elk vakje van die array zowel een naam, een emailadres, een nickname en een nummer kunnen staan.
In regel 12 wordt een variabele ‘nr’ gedeclareerd die we als teller gebruiken en ook gebruiken we die in het stukje 20 t/m 25 om aan te geven in welk vakje van de array de gegevens bewaard zullen worden. De variabele ‘welk’ wordt gebruikt om een vakje uit het array te kiezen.

Opdr6_b
Schrijf zelf een programma dat een naam, en een telefoonnummer kan vragen aan vier personen.Als er vier zijn ingevuld moet je er een kunnen opvragen. De variabelen die je gebruikt moeten logisch zijn dus een telefoonnummer heet geen X of Z.

```
program grafisch_leuk;
uses crt;

type stijl = record
   achtergrond:integer;
   letters    :integer;
   text      :string;
   geluid     :integer;
   end;

stijlen = array[1..3] of stijl;

var welk: stijlen;
   keuze,teller:integer;
begin
clrscr;
teller:=1;
for teller:= 1 to 3 do
begin
writeln('geef me een achtergrondkleur in turbopascal code : ');
readln(welk[teller].achtergrond);
writeln('Geef me een textkleur in tp code              : ');
readln(welk[teller].letters);
writeln('Welke text had je daar in willen hebben?       : ');
readln(welk[teller].text);
writeln('En welk geluid? een getal van 200 tm 2000      : ');
readln(welk[teller].geluid);
clrscr;
end;

writeln('welke wil je zien en horen?  : ');
readln(keuze);
clrscr;
textbackground(welk[keuze].achtergrond);
textcolor(welk[keuze].letters);
clrscr;
writeln(welk[keuze].text);
sound(welk[keuze].geluid);
delay(1000);
nosound;
readln;
end.
```

Opg6_c

Type het bovenstaande programma over en verander het zo dat er nog gevraagd wordt naar een schermpositie waar de text moet komen (vraag x en y apart).

En naar de duur van het delay, dus hoe lang moet dit gaan piepen.

Vanuit een programma files bewaren en openen

Dit hoofdstuk is niet zo moeilijk maar wel heel leuk. Eindelijk leer je eens hoe je een programma kan maken waarmee je een file naar de harde schijf kan schrijven. Uiteindelijk maak je een simpel soort kladblok.

```
program pietje;
uses crt;
var bestand: Text;
    alles:string;

begin
       alles:='c:\pietje.txt';

      Assign(bestand,alles);
      Rewrite(bestand);
      clrscr;
      Writeln(bestand,'pietjepietjepietjepietje');
     Close(bestand);
  writeln('de text is opgeslagen als :pietje.txt');
  write('het was me weer een waar genoegen, aju!! enter om af te
sluiten.');
  readln;
end.
```

Typ over en zie dat er een textfile met de naam pietje op je schijf gezet wordt als je het programma runt.
Als er al een file met de naam pietje was is die file nu overschreven.

Opg7_a
Verander bovenstaand programma zo dat je zelf een text mag invullen dus eerst een text uitlezen met behulp van readln(text) en die schrijf je dan in de file met writeln(bestand,text).
Bij **opgave 7a** worden de volgende functies gebruikt die nieuw zijn:

Assign

Syntax:

```
Assign(bestand,'c:\koekjes\kletskop.txt');
```

bestand is hierbij een variabele die in het programma gebruikt zal worden om te communiceren met de file kletskop.txt, die in de directory c:\koekjes zit.

Textfiles

Textfiles kennen we uit windows met de extensie .txt.
Als we bovenaan het programma dus de variabele bestand declareren als text
Var bestand:text;
Dan wordt dat begrepen door TurboPascal. Textfiles bestaan uit lange reeksen chars. Dus TurboPascal ziet zo'n textfile, bijvoorbeeld een brief aan je geliefde, niet als een stapel regels maar als een enorm aantal letters en spaties, de chars.
Een textfile zal dus ook letter voor letter gevuld en gelezen worden.

Rewrite
Rewrite wordt gebruikt om een bestand te maken en te openen. Met Rewrite maak je dus een file aan op de schijf. Een bestaande file met dezelfde naam wordt op dat moment wel overschreven!!
Syntax:

Rewrite(bestand);

bestand is hier diezelfde bestand als bij Assign is gebruikt. Het commando Rewrite 'kijkt' bij Assign waar die file gemaakt moet worden. Natuurlijk moet de direktory waar Rewrite moet gaan schrijven wel bestaan. Het maken van directory's komt later aan de orde.

Close

Als er teveel bestanden tegelijkertijd geopend zijn kan er een foutmelding komen.
Na gebruik kan je de file weer netjes afsluiten met Close.
Syntax:

Close(bestand);

Bestand is weer dezelfde file die we bij Assign benaderd hebben en bij Rewrite hebben gemaakt.

Writeln

En het gebruik van Writeln in een ‘assignde‘ file (bestand) ;
Writeln(bestand,text)
De standaard uitvoer van Writeln is zoals we gewend zijn het beeldscherm.
Je kan Writeln ook een string in een file laten schrijven.
Syntax:

```
Writeln(bestand,text);
```

bestand is de al bekende file, text is een variabele die we bijvoorbeeld met behulp van Readln(text) hebben gevuld.

Opg7_b
Schrijf met behulp van het volgende stukje een programma dat meerdere regels text in een file wegschrijft totdat je een lege regel typt en op de <enter> -knop duwt.

```
repeat
        readln(text);
        Writeln(F, text);
        until text='';
        Close(F);
```

Opg7_c
Breng weer een verandering aan in het programma;
Zorg er voor dat je in het begin de vraag krijgt hoe het bestand moet heten en waar het opgeslagen moet worden. Deze string bv. C:\poekie.txt kan je uitlezen met readln en dan met assign aan de variabele bestand ‘hangen’.

Opg7_d
Natuurlijk is het veel mooier om na het schrijven pas een naam aan het bestand te geven
Verander het programma van **opg7_c** zodanig dat er eerst om een tekst gevraag wordt (meerdere regels) en dan vraagt om een naam en pad.
Regel het zo dat de extensie automatisch .txt wordt.
Even een stukje stoeien met strings en die aan elkaar plakken. Maak gebruik van een tijdelijke file!
Bij **opgave 7d** zijn nieuw;

Rename

Deze opdracht kan gebruikt worden om een bestand een andere naam te geven.
Het bestand moet dan wel Assigned zijn!!
Syntax:

Rename(F,'c:\koekjes\janhagel.txt');

F is nu hopelijk wel bekend. De file kletskop.txt krijgt nu de naam janhagel.txt.
Met het volgende programma kan je de text die in een bestand pietje.txt zit uitlezen.
Schrijf over en probeer het.

```
1.  Program lees_pietje;
2.  Uses crt;
3.  Var bestand:text;
4.  Ch      :char;
5.  begin
6.  clrscr;
7.  Assign(bestand,'c:\pietje.txt');
8.  Reset(bestand);
9.  while not Eof(bestand) do
10. begin
11. Read(bestand, Ch);
12. Write(Ch)
13. End;
14. Readln;
15.       End.
```

Opg7_g
Maak een programma dat vraagt welke textfile je wilt inlezen. Als de filenaam en het pad zijn opgegeven moet de text op het scherm getoond worden.

Opg7_f
Voeg Opg7_d en Opg7_e samen.
Verander je eigen notepad nu zo dat je eerst een menu krijgt waar de mogelijkheden opvragen en zelf schrijven aanwezig zijn.
Bij het opvragen van een file is het niet nodig om ook dingen in die file te kunnen veranderen of te navigeren binnen de text.

Opgave 1c, de nieuwe procedures

Reset

Deze opdracht opend een file voor een reeds bestaand bestand. Dit bestand wordt, anders dan bij Rewrite niet overschreven maar alleen geopend.
Syntax:

```
Reset(bestand);
```

bestand is die ene file weet je nog wel? Natuurlijk moet die file wel weer assigned zijn. Ook Reset leent de informatie van Assign om te weten welke file geopend moet worden.

Read

Read is natuurlijk dezelfde Read als gebruikt wordt om van ‚standaard invoer, het scherm te lezen. Hij, of zij kan echter ook gebruikt worden om uit een file te lezen.Deze file moet natuurlijk wel ‘assigned’ zijn.
Syntax:

```
Read(bestand,letter);
```

bestand kennen we nu wel. Letter is een variabele die bovenaan gedeclareerd is als:
Var letter:Char;

Eof

Door de procedure Rewrite word in een nieuw bestand een end of file-teken gezet. End of File is dus een tekentje waar de inhoud van het bestand voor geplaatst wordt.

Bijvoorbeeld:

```
 while not Eof(bestand) do
 begin
Read(bestand,letter);
Upcase(letter);   {maak er een hoofdletter van}
Write(bestand,letter);
End;
```

Hoofdstuk 8 zoeken naar een bestand.

In dit hoofdstuk ga je leren hoe je een bestand kan vinden met je zelf geschreven programma.
Je gaat dit toepassen bij het maken van een loginscherm met wachtwoord.

```
1       Program bestaat_de_file;
2       uses crt,dos;
3       var bestand:file;
4       filenaam:string;
5       bestaat: integer ;

6       begin

7           clrscr;
8           writeln('Gimme een filenaam met pad: ');
9           readln(filenaam);
10          assign(bestand,filenaam);
11          {$I-}
12          reset(bestand);
13          close(bestand);
14          {$I+}
15          bestaat:=ioresult;
16                      if bestaat=0 then
17                        begin
18                        writeln('de file ',filenaam,' bestaat.');
19                        readln;
20                  end
21                      else
22                         begin
23                         writeln('De file ',filenaam,' bestaat niet.');
24                         readln;
25                         end;
26      end.
```

Regel 11 **{$I-}** = uitschakelen I/O check
Regel 14 **{$I+}** = inschakelen I/O check
Deze rare dingen worden gebruikt tijdens het compilen van een geschreven programma.
Als er tijdens het runnen een I/O foutmelding wordt verwacht.
Zorgt het eerste stukje syntax ervoor dat de compiler niet blijft zeuren om verandering omdat er iets niet klopt. En met het laatste stukje syntax kan je er weer voor zorgen dat de compiler de rest van het programma wel even nakijkt op fouten.

IOResult
Deze functie wordt meestal gebruikt bij het checken van een file. Omdat er toch iets niet goed gaat wil je dan toch weten wat er niet goed gaat.
De integer functie IOResult geeft een waarde die het verloop van de meest recente I/O aktie weergeeft. Als IOResult de waarde 0 heeft is de aktie gelukt.
Als er iets fout gaat krijgt deze functie een andere waarde.
Bv: 2 bestand niet gevonden
pad niet gevonden
3 teveel bestanden open (maximum is 15)
4 acces denied
Syntax:(van de twee laatste samen)

```
{$I-}
assing(bestand,c:\pietje.txt);
reset(bestand);
{$I+}
bestaat:=ioresult;
if bestaat=0 then
begin
writeln('Joepie de file pietje bestaat');
```

Opg8_a
Hierboven staat een programma dat kijkt of een file bestaat. Verander het zo dat ie kijkt of een bepaalde file bestaat die je wawo.txt hebt genoemd en die het pad c:\ heeft.
Als die file niet bestaat, moet het programma dat melden met een waarschuwing dat de computer niet beveiligd is. Dus het moet in het programma verwerkt zitten dat ie altijd op zoek gaat naar de file c:\wawo.txt.

Opg8_b
Verander het programma van opg8_a zo dat er als er geen c:\wawo.txt bestaat, er de mogelijkheid wordt geboden om er zelf een te maken.
De tweede keer dat je het programma draait moet ie dus aangeven dat er wel een c:\wawo.txt is.

```
while not Eof(bestand) do
          begin
           Readln(bestand,wawo);
            if wawo=poging then
            writeln('come in pall')
            else
            writeln('nee dus');
```

```
end;
```

Met behulp van bovenstaand stukje kan je dus het wachtwoord lezen en vergelijken. Natuurlijk moet je het programma dat je bij 8_b hebt gemaakt ook gebruiken.

Opg8_c

Maak een programma dat een wachtwoord vraagt als er een wawo.txt bestaat. Als dat niet bestaat moet dat eerst aangemaakt worden. Bij de 2e keer runnen van het programma moet de juiste file worden uitgelezen.

Hoofdstuk 9 nog meer bestanden uitlezen

```
1       Program zoek_woord;
2       uses crt;
3        var woord,woord2:string;
4                bestand:text;
5           x:integer;
6        begin
7        clrscr;
8        x:=0;
9        assign(bestand,'c:\zoeken.txt');
10       reset(bestand);
11       writeln('mag ik een woord');
12       readln(woord);
13       while not eof(bestand) do
14       begin
15           readln(bestand,woord2);
16           if woord=woord2 then
17           begin writeln('Joepie, ik heb het gevonden');
18           x:=1;
19           end;
20       end;

21        if x=0 then
22        writeln('Sorry niet gevonden');
23        readln
24        end.
```

Maak een file c:\zoeken.txt en zet daar een aantal woorden in onder elkaar b.v.

Inhoud c:\zoeken.txt:
Goeiendag
Kip
Geit
Paard
Koe
Nu
Toen
Gisteren
Blij
Gein
Niks

Type de lijst zo over en dus op elke regel maar één woord.
Schrijf dan het bovenstaande programma over en run het.

Op deze manier zoek je dus in een file naar een bepaald woord, dit werkt op deze manier alleen als je de woorden onder elkaar zet omdat het programma regel voor regel leest.

Het uitlezen van meerdere gegevens tegelijk

Als je gebruik maakt van de record opdracht kan je een eigen type maken waar meerdere gegevens inzitten.

```
type
wachtwoorden=record
naam    :string;
wachtwoord:string;
end;
```

Nu heb je een type gemaakt dat wachtwoorden heet. Een variabele, die je nog moet declareren bij var heeft dus eigenlijk twee onderdelen. Er zit een ‘naam’ in en een ‘wachtwoord’.

```
1       program wachtwoord_maken;
2       uses crt,dos;
3       type
4       wachtwoorden=record
5       naam    :string;
6       wachtwoord:string;
7       end;

7       var tekst :string;
8       bestand :file of wachtwoorden;
10          gegevens:wachtwoorden;
11          omvang:integer;
12      begin
13          assign(bestand,'c:\wawo.duc');
14          rewrite(bestand);
15          writeln('Gimme je naam : ');
16          readln(gegevens.naam);
17                      repeat
18                      writeln('Gimme het wachtwoord dat je wilt
gebruiken : ');
19                      readln(gegevens.wachtwoord);
20                      writeln('Nogmaals het wachtwoord :');
21                      readln(tekst);
22                      until tekst=gegevens.wachtwoord;
23                      omvang:=filesize(bestand);
24                      seek(bestand,omvang);
25                     write(bestand,gegevens);
26          writeln('Je gegevens zijn opgeslagen!');
27          readln
28      end.
```

In regel 2 zetten we nu ook de dos-unit bij de units die we gaan gebruiken.
In regel 3 t/m 7 wordt het nieuwe type wachtwoorden gemaakt.
Zie dat bij var, het declareren van de variabelen, de variabele gegevens wordt gedeclareerd als zijnde van type wachtwoorden. Een variabele van dat type bevat dus twee dingen: Een naam en een wachtwoord.
Die zullen we later in het programma gebruiken als gegevens.naam en gegevens.wachtwoord.
Regel 12 en 13 laten zien dat er een nieuw bestand wordt gemaakt op C:\ het bestand wawo.duc. Het maakt niet uit hoe dat bestand heet en de extensie maakt ook niet uit, drie letters is wel zo netjes.
In Regel 16 wordt de ingegeven naam dus gezet in gegevens.naam. In regel 19 wordt het wachtwoord gezet in gegevens.wachtwoord.
De regels 17 t/m 21 zijn binnen een repeat-lus gezet om er voor te zorgen dat iemand geen typefout maakt, en zijn wachtwoord goed onthoudt. In het echte

leven verschijnen wachtwoorden namelijk in de vorm van sterretjes op het beeldscherm zodat een verkeerde toetsaanslag niet gecontroleerd wordt.
Regel 23 bevat het commando ‘filesize(bestand)’ Dit commando zoekt naar de hoeveelheid variabelen die in het bestand aanwezig zijn, omdat dit bestand met rewrite net vers is gemaakt zal dat altijd op nul staan. Nu weet het programma hoeveel er al in het bestand staat. Dat moeten we weten omdat we er nog iets bij willen schrijven (naam en wachtwoord).
In regel 24 zetten we met de opdracht seek(bestand,omvang) de cursor min of meer in het bestand op de plaats achter het laatste wat er in het bestand is geschreven. Omdat ‘omvang’ nog op 0 staat zal dat dus vooraan in de file zijn.
Zo kan in regel 25 de variabele ‘gegevens’ (die bestaat uit gegevens.naam en gegevens.wachtwoord) achter in de file geschreven worden.

Type het programma wachtwoord_maken over en run het. Ga daarna op zoek naar de file wawo.duc en kijk met behulp van kladblok of in dos met behulp van type wat er nu eigenlijk in staat.

Opg9_a
Als je zeker weet dat het bestand op je harde schijf staat, kan je regel 14 veranderen in reset(bestand). Er zal dan ook al een naam en een wachtwoord in het bestand staan. Seek zal automatisch op zoek gaan naar het einde en er een nieuwe naam en wachtwoord bij zetten.
Doe dit en kijk met type of kladblok wat er gebeurt.

Opg9_b
In hoofstuk 8 heb je het programma bestaat_de_file gemaakt. Dat programma controleert of er een bepaalde file bestaat. Gebruik dat programma en Opg9_a om een programma te schrijven dat kijkt of er een wawo.duc aanwezig is of niet. Zo ja geef een wachtwoord en een naam op. Zo nee maak de file en vul een wachtwoord in.

```
1.  program wachtwoord_lezen;
2.  uses crt,dos;
3.  type
4.          wachtwoorden=record
5.          naam    :string;
6.          wachtwoord:string;
7.  end;

8.  var tekst :string;
9.  bestand :file of wachtwoorden;
10. gegevens:wachtwoorden;
11. omvang,keus:integer;
```

```
12. begin
13.         assign(bestand,'c:\wawo.duc');
14.         reset(bestand);
15.         omvang:=filesize(bestand);
16.         writeln('het bestand bevat',omvang,' keer gegevens.');
17.         writeln('kies welke gegevens je wilt zien');
18.         readln(keus);
19.         seek(bestand,keus);
20.         read(bestand,gegevens);
21.         clrscr;
22.         writeln('naam is: ',gegevens.naam);
23.         writeln('wachtwoord is: ',gegevens.wachtwoord);


24. readln
25. end.
```

Opg9_c
Als je Opg9_b gemaakt hebt dan kan je een heleboel wachtwoorden en namen invullen alleen gebeurt er niets mee. Type bovenstaand programma over en zie dat je gegevens uit wawo.duc kan uitlezen.
Als je die gegevens op het beeldscherm kan tonen, kan je natuurlijk ook het programma de opdracht geven om bij een bepaald getal (dat je verkrijgt met de seek-opdracht) het juiste wachtwoord en de juiste naam te krijgen.
De eindopdracht voor Hoofdstuk 9 is het schrijven van twee programma's: Een programma is voor de systeembeheerder die wachtwoorden kan invoeren en uitlezen.
Het andere programma is voor een gebruiker die van de systeembeheerder een getal, een naam en een wachtwoord krijgt. Bij juist invullen bij het tweede programma krijgt een gebruiker een melding dat hij naar binnen mag of een melding dat hij naar de systeembeheerder moet voor een nieuw wachtwoord.
De interface voor het gebruikersprogramma moet dus de volgende vragen stellen.
Wat is uw nummer? :
Wat is uw naam? :
Gimme uw wachtwoord:

Hoofdstuk 10:

In dit hoofstuk ga je een kleine database maken. De database kan je gebruiken als een adresboek voor emailadressen. Je hebt eerst gezocht door te zoeken met een nummer dat gekoppeld is aan 'filesize' je kan echter ook gewoon op een naam zoeken. Je geeft opdracht om te zoeken naar een naam.

```
1procedure zoeken;
2var naam:string;
3  a  :integer;
4begin
5       clrscr;
6       assign(bestand,'c:\meel.dat');
7       reset(bestand);
8       writeln('wiens adres wil je zoeken');
9       readln(naam);
10      while not eof(bestand)
11      do
12              begin
13              a:=1;
14              read(bestand,persoons_gegevens);
```

```
15              if naam=persoons_gegevens.naam then
16                      begin
17                      a:=2;
18                      with persoons_gegevens do
19                              begin
20                              writeln('naam          :',naam);
21                              writeln('nickname      :',nickname);
22                              writeln('emailadres    :',emailadres);
23                              writeln('extra gegevens :',extra_gegevens);
24                              end;
25                      end;
26              end;
27      if a=1 then
28      writeln('helaas die naam heb ik niet in mijn bestand')
29      end;
```

Deze procedure zoeken kan je gebruiken in je programma.
In regel 3 declareer ik de variabele a die ik alleen gebruik om in regel 27 te bepalen of er een melding moet komen over niet gevonden gegevens.
In regel 10 geef ik aan dat het programma tot het eind van het bestand moet zoeken naar de opgegeven naam. Bijvoorbeeld piet. Nu zal als piet gelijk is aan een 'persoons_gegevens.naam' het programma van regel 18 tot en met regel 23 alle persoonsgegevens op het scherm tonen.
Natuurlijk heb ik in het begin van het programma een aantal dingen als volgt gedeclareerd.
Ook heb ik natuurlijk mijn eigen type verzonnen.

```
type gegevens =record
naam:string;
nickname:string;
emailadres:string;
extra_gegevens:string;
end;

var
persoons_gegevens:gegevens;
bestand:file of gegevens;
```

Opg10_a
Maak nu met deze gegevens een programma waar je emailadressen mee op kan slaan en natuurlijk ook op kan vragen.
In het begin van het programma moet je een keuzemenu krijgen waar je de keuze hebt uit 1.het invoeren van gegevens 2. het opvragen van gegevens en 3.het sluiten van het programma.
Nadat je iets hebt ingevoerd moet je weer uitkomen bij het menu om weer een keuze te hebben tussen de drie mogelijkheden. Het programma moet dus doorgaan totdat je voor 3 kiest.
Als je de eerste keer met het programma werkt moet er gecontroleerd worden of c:\meel.dat bestaat en als die niet bestaat moet die natuurlijk automatisch aangemaakt worden.
De volgende keer moet de meel.dat gevonden en niet overschreven worden.

11 units LUIHEID BLIJHEID

Units, ze maken het leven gemakkelijker.
Tot nu toe heb je in je programma's gebruik gemaakt van Units zonder dat je misschien wist wat dat nou eigenlijk inhield.
Met het zinnetje : Uses crt,dos;
Geef je de compiler de opdracht om naar die Units te kijken als bepaalde opdrachten gegeven worden die anders niet begrepen worden. Bijvoorbeeld de functie Clrscr wordt niet automatisch door de compiler begrepen. In de Unit crt, staat beschreven wat de compiler moet doen.
Deze units kan je ook zelf makkelijk maken.
Ik heb al eerder aangehaald dat goede programmeurs lui moeten wezen. Wat je kan láten doen moet je niet zelf gaan zitten typen.
Je zou bijvoorbeeld een unit 'adres' kunnen maken die als je de opdracht 'adres;' in je programma zet, je automatisch in je programma je adres op het beeldscherm laat zien krijgt.
Je kan een heleboel functies in een unit proppen en zo de veel door jou gebruikte functies vereenvoudigen.
Het is even een beetje werk maar daarna ben je een echte crack die zijn eigen Units meeneemt als ie ergens gaat programmeren.
Een voorbeeld:

```
1       Unit feest;

2       interface

3       procedure pp(d,h:integer);
4       {pp staat voor piep. vraagt twee variabelen, een voor toonhoogte}
5       {en een voor duur van het geluid werkt met systeem luidspreker  }
}

6       implementation

7       uses crt;

8       procedure pp(d,h:integer);
9       begin
10      sound(d);
11      delay(h);
12      nosound;
13      end;

14      end.
```

In deze unit, met de naam feest, staat een functie beschreven die pp heet.
In regel 1 staat het woord Unit, de compiler begrijpt nu dat als je deze .pas file compileerd dat ie er geen exe, maar een tpu file van moet maken. Tpu staat voor turbo-pascal unit.
Regel 2 bevat alleen het woord ‘interface’ dit geeft aan dat er nu vertelt kan worden welke functies deze unit bevat. In dit geval is dat er maar één namelijk de procedure pp. Er moet ook vertelt worden op welke manier deze functie aangeroepen moet worden, namelijk met twee variabelen van het type integer.
Regel 4 en 5 bevatten een korte uitleg van de functie pp zodat je later boven in de files de informatie kan vinden over de functies die de unit allemaal bevat. Vooral als je een hele grote unit maakt met heel veel functies, moet je natuurlijk niet de weg kwijt raken.
Regel 6 bevat alleen het woord implementation. Dit geeft aan dat hierna de uitwerking van de functies komen.
Regel 7: Jawel deze unit heeft weer andere units nodig om gecompileerd te worden, dat geef je op de ouderwetse manier aan met ‘uses’.
In regel 8 tm 13 schrijf ik gewoon een procedure die ik anders ook in een programma had kunnen schrijven.
Op regel 14 eindigt het hele verhaal ook weer lekker ouderwets met end..

In plaats van al die procedures elke keer weer in een programma erbij te moeten uitwerken, kan ik op deze manier met alleen het woordje feest volstaan.

Bijvoorbeeld in het volgende programma.

1. **Program boer_er_ligt_een_kip_in_het_water;**
2. **Uses crt,feest;**

3. **Begin**
4. **Pp(**264**,500);**
5. **Pp(**297**,500);**
6. **Pp(**330**,500);**
7. **Pp(**352**,500);**
8. **Pp(**395**,500);**
9. **Pp(**352**,500);**
10. **Pp(**330**,500);**
11. **Pp(**297**,500);**
12. **Pp(**264**,500);**
13. **End.**

Opdracht:
Maak de unit en compile die tot een tpu. Kijk in de direktory bij je andere opdrachten of ie er inderdaad bij zit.
Maak dan het programma boer_er_ligt_een_kip_in_het_water; en run het.

Nog meer Units:
Voeg text bij aan de unit feest totdat het er zo uitziet.

```
1.  Unit feest;

2.  interface
3.  procedure tk(x,y,t,a:integer;wat:string);
4.  {tk staat voor text in kleur heeft x,en y coordinaat nodig}
5.  { en textachtergrond en textkleurcode}
6.  procedure pp(d,h:integer);
7.  {pp staat voor piep. vraagt twee variabelen, een voor toonhoogte
    }
8.  {en een voor duur van het geluid werkt met systeem luidspreker
    }

9.  implementation
10. uses crt;
11. procedure tk(x,y,t,a:integer;wat:string);

12. begin
13. gotoxy(x,y);
14. textbackground(a);
15. clreol;
16. textcolor(t);
17. write(wat);
18. textbackground(0);
19. clreol;
20. end;

21. procedure pp(d,h:integer);
22. begin
23. sound(d);
24. delay(h);
25. nosound;
26. end;

27. end.
```

Opdracht:
Tiep het volgende over en run het:

```
1.  Program gimmie;
2.  Uses crt,feest;

3.  Begin
4.  Clrscr;
5.  Pp(400,200);
6.  Tk(5,10,0,14,'gimmie');
7.  Pp(400,200);
8.  Tk(35,20,4,14,'gimmie');
9.  Pp(2000,200);
10. Tk(50,2,6,12,'gimmie');
11. Pp(700,200);
12. Tk(3,22,0,11,'gimmie');
13. End.
```

Het kan zijn dat je de feest-unit ook nog even in je werkomgeving 'open moet laten staan' anders zoekt het programma in de unit direktory van turbopascal. Je kan natuurlijk ook de feest.tpu naar die direktorie copieren. Het programma gimmie heeft in het uitvoerende gedeelte alleen nog maar het voor ons herkenbare clrscr; staan. De andere codes zijn overgenomen door de zelfbedachte unit.

Opdracht:
Gebruik het principe van opdracht 1 b. Maak een unit of breidt de unit feest uit, waarin deze 'ga weg' procedure voorkomt. Zorg er nu voor dat er in plaats van 'ga weg', de text Hallo… komt te staan. Op de plaats van … moet je dan in je programma een naam kunnen opgeven.
Hoe de Unit eruit moet komen te zien mag je zelf weten. Het programma dat je schrijft moet er zo uitzien.

```
1.  Program hallo_boer;
2.  Uses crt,feest;
3.  Begin
4.  Ha('boer');
5.  End.
```

Als je dit verhaal runt moet er dus de text 'hallo boer' in de vier hoeken van je scherm verschijnen en dat moet in 5 stappen naar het midden van het scherm 'lopen'.

Nog meer Units.

In een Unit kan je ook functies onderbrengen.

Je kan dus je eigen wis-unit maken waarin je de functies optellen, aftrekken, delen, wortelen, machtverheffen enzo onderbrengt.

```
1.  Unit wis;

2.  Interface
3.  Function plus(a,b:integer):integer;
4.  {telt twee opgegeven variabelen op}
5.  function min(c,d:integer):integer;
6.  {trekt de laatste variabele van de eerste af}

7.  implementation
8.  Uses crt;

9.  Function plus(a,b:integer):integer;
10. begin
11. plus:=(a+b);
12. end;

13. function min(c,d:integer):integer;
14. begin
15. min:=c-d;
16. end;
17. end.
```