4.1 Inleiding

In het vorige hoofdstuk heb je gelezen dat het geheugen van een computer verdeeld is in adressen en dat op die adressen gegevens kunnen staan. Deze gegevens staan binair genoteerd. Als we programmeren worden de gegevens in het geheugen van de computer opgeslagen. Iedere byte die opgeslagen wordt, heeft een eigen adres. Deze bytes worden bij de uitvoering van een programma vaak weer tevoorschijn gehaald, bewerkt en teruggezet, of gewist.

Het zou wat lastig worden als je als programmeur zelf zou moeten bijhouden op welk adres je een bepaald gegeven gezet hebt als je het later weer wilt ophalen. Die taak voert de compiler voor je uit. De compiler werkt hiervoor nauw samen met het besturingssysteem, zoals bijvoorbeeld Windows of Linux. Er wordt niet alleen bijgehouden waar gegevens neergezet zijn, er wordt ook bijgehouden om wat voor soort gegevens het gaat. Is het een getal? Is het een letter? Is het een zin of een heel tekstblok?

De programmeur geeft aan een bepaald gegeven een naam en vermeldt daarbij om wat voor soort gegeven het gaat. Free Pascal doet de rest. De naam die een bepaald gegeven krijgt, noemen we een variabele en de soort noemen we een type. Als we zouden beschikken over een type dat GETAL heet en we willen het resultaat van een optelling bewaren, dan zou je een variabele TOTAAL kunnen opgeven (declareren) met als type-aanduiding GETAL.

Free Pascal levert een groot aantal standaardtypen. Het is echter ook mogelijk zelf een type te maken. In dit hoofdstuk worden eerst de standaardtypen behandeld en daarna wordt gedemonstreerd hoe je zelf typen kunt definiëren. Bij de programmavoorbeelden zal vaak een regelnummering meelopen. Onder het voorbeeld staat dan per regelnummer (of per groep regelnummers) wat er op die regels in het programma gebeurt. Als er op zo'n regel een nieuwe instructie aan de orde is of een instructie die een nadere toelichting vereist, dan staat er in de kantlijn een nummer dat verwijst naar een alinea in de toelichting.

4.2 Het type Byte

Zoals de naam al aangeeft, is Byte een type dat één byte (dus acht bits) in het geheugen in beslag neemt. Dit type is bestemd voor het opslaan van kleine getallen.

In het vorige hoofdstuk hebben we gezien dat in de acht bits van een byte maximaal de waarde 255 geschreven kan worden. Het grootste getal dat het type Byte kan bevatten, is dus 255. In het volgende programmaatje is de variabele GETAL van het type Byte:

```
PROGRAM TYPE_1;
USES CRT;

VAR
   GETAL: Byte;

BEGIN
   ClrScr;
   GETAL := 65;
   GotoXY(20,10);
   Write('De waarde van GETAL is ',GETAL);
   GETAL := GETAL + 32;
   GotoXY(20,11);
   Write('Na optelling is de waarde van GETAL ',GETAL);
   Readln
END.
```

Regels: Omschrijving:

1 De naam van het programma is TYPE_1.
2 Gebruik uit de programmabibliotheek de unit CRT.
[1] 3-4 Declareer een variabele van het type Byte en noem deze GETAL.
5 Start het programma.
6 Veeg het scherm schoon.
[2]7 Geef de variabele GETAL de waarde 65.
8 Ga naar positie 20 op regel 10.
[3]9 Plaats een zin en de waarde die in GETAL staat op het scherm.
[4]10 Tel 32 op bij de waarde die in GETAL staat.
11 Ga naar positie 20 op regel 11.
12Schrijf een zin en de nieuwe waarde die in GETAL staat naar het scherm.
13Wacht met de beëindiging van het programma tot op de Enter-toets is gedrukt.

Op de volgende pagina vind je de toelichting op dit programmaatje.

Toelichting:

[1]Een variabele wordt gedeclareerd onder het beschermde woord VAR. Een beschermd woord wil zeggen dat dit woord niet elders in de programmacode voor een ander doel gebruikt mag worden. Je mag als programmeur een variabele dus niet VAR noemen. Na het woord VAR volgt een variabelenaam of een aantal variabelenamen. Na de naam wordt een dubbele punt gezet met daarachter de aanduiding van het type. Als je meerdere variabelen van hetzelfde type wilt declareren, worden ze met een komma van elkaar gescheiden. Als je dus in het programma TYPE_1 een tweede variabele van hetzelfde type nodig zou hebben, zouden de regels er als volgt uit kunnen zien:

```
VAR
   GETAL, GETAL_2: Byte;
```

[2]Met het teken := wordt een waarde in de variabele GETAL gezet. Dit teken wordt de toewijzingsoperator genoemd. Een operator is een symbool of een teken waarmee aangegeven wordt welke bewerking uitgevoerd moet worden. In hoofdstuk 13 wordt het gebruik van operatoren uitgelegd met een toelichting op hun functie en de manier waarop ze gebruikt worden. Als er aan GETAL een grotere waarde dan 255 toegewezen wordt, dan constateert de compiler een fout en krijg je de volgende boodschap te zien:

```
Error 76 Constant out of range
```

Dit wil zeggen dat de waarde die je aan GETAL wilt toewijzen buiten het waardebereik van GETAL ligt, want in één byte kan maximaal de waarde 255 geschreven worden.

[3]Met het aanroepen van de procedure Write wordt een zin op het scherm geplaatst. Let er op dat de zin die tussen de haakjes geplaatst wordt tussen aanhalingstekens geplaatst moet worden. Na de zin staat een komma, gevolgd door het woord GETAL. Omdat de aanhalingstekens hier ontbreken, weet de procedure Write dat het hier om een variabele gaat. De procedure Write zal de waarde die in GETAL staat naar het scherm schrijven.

[4] Hier kunnen we zien op welke wijze we een variabele kunnen bewerken. In dit geval wordt bij de waarde die in GETAL staat 32 opgeteld, en vervolgens wordt de nieuwe waarde in de variabele GETAL gezet. In plaats van de uitdrukking "variabele" wordt ook vaak de uitdrukking "veld" genoemd. Je hebt het

dan over het veld GETAL.

4.3 Het type Char

Evenals het type Byte is het type Char 1 byte groot. Het type Char is echter niet bestemd om getallen weer te geven, maar letters en leestekens. Char is een afkorting van het Engelse woord "character", dat letter of leesteken betekent.

De manier waarop letters en leestekens weergegeven worden, is een verhaal apart. In bijlage 7 staat de ASCII-tabel. ASCII is een afkorting van "American Standard Code for Information Interchange". Vertaald wil dit zeggen: "Amerikaanse standaardcode voor informatie-uitwisseling". In de tabel is aan alle letters en leestekens een nummer toegewezen. Als je de letter "A" in een variabele van het type Char plaatst, wordt het getal 65 in het geheugen opgeslagen. Als een variabele als type Char gedeclareerd is, wordt er niet een getal, maar een letter afgebeeld. Het volgende programmaatje zal dit verduidelijken:

```
PROGRAM TYPE_2;
USES CRT;
VAR
   LETTER_1, LETTER_2: Char;
   GETAL             : Byte;

BEGIN
   ClrScr;
   GETAL := 65;
   LETTER_1 := 'A';
   LETTER_2 := Chr(GETAL);
   GotoXY(20,10);
   Write('In LETTER_1 staat:',LETTER_1,
        ' In LETTER_2 staat:',LETTER_2 );
   GETAL := GETAL + 32;
   LETTER_2 := Chr(GETAL);
   GotoXY(20,11);
   Write('In LETTER_1 staat:',LETTER_1,
          ' In LETTER_2 staat:',LETTER_2 );
   Readln
END.
```

Regels: **Omschrijving**:

[1]3-5 Declareer de benodigde variabelen.
7 Veeg het scherm schoon.
8 Zet de waarde 65 in GETAL.
[2] 9 Zet de letter "A" in het veld LETTER_1.
[2]10 Zet in LETTER_2 de letter die in de ASCII-tabel het nummer heeft dat overeenkomt met de waarde die in GETAL staat.
[3]11-13Zet op de twintigste positie van de tiende regel van het scherm een zin en de waarden die in LETTER_1 en in LETTER_2 staan.
[4]14Verhoog GETAL met 32.
[4]16-18 Zet de eerder geschreven zin nog eens op het scherm.

Toelichting:

[1] Hier worden drie variabelen gedeclareerd. LETTER_1 en LETTER_2 zijn beide van het type Char. De variabele GETAL van het type Byte wordt ook hier weer gebruikt.

[2] In LETTER_1 zetten we de hoofdletter "A". We doen dit met behulp van de toewijzingsoperator en een letter tussen aanhalingstekens. In plaats van LETTER_1 := ’A’ hadden we ook kunnen zeggen LETTER_1 := #65. Dit is een tweede manier om een letter toe te wijzen. Het hekje fungeert hier als operator om aan te geven dat het hier om een letter of leesteken gaat. Met behulp van de functie Chr zetten we de letter in LETTER_2. Deze derde manier om een letter toe te wijzen, zet een gegeven waarde om in de letter die volgens de ASCII-tabel bij deze waarde hoort. Doordat we tussen haakjes de variabele GETAL zetten, wordt in LETTER_2 de letter gezet die in de ASCII-tabel een overeenkomstige waarde heeft. In GETAL stond de waarde 65. Als we in de ASCII-tabel kijken, dan zien we dat letter 65 de hoofdletter "A" is.

[3] We zien hier dat de komma het scheidingsteken tussen tekst en variabelen binnen de Write-procedure is.

[4] Als we nu GETAL met 32 verhogen en we zetten de regel nog eens op het scherm, dan zie je dat de hoofdletter "A" veranderd is in een kleine letter "a". Alle kleine letters hebben een nummer dat 32 hoger is dan het nummer van de bijbehorende hoofdletter.

4.4 Het type Array

Het woordenboek geeft als vertaling van het Engelse woord "array": "stoet" of "rij". Dit geeft heel duidelijk aan waar het bij het type Array om gaat. Eerst zal in deze paragraaf een algemene toelichting op het begrip array’s gegeven worden, later volgt dan een toepassing. Een array wordt altijd gedeclareerd als rij gegevens van hetzelfde type. Als je een array definieert, reserveer je een opeenlopend aantal "blokjes" (bytes) in het geheugen om gegevens van hetzelfde type in op te slaan. Bij de declaratie moeten we ook opgeven hoe groot de array wordt. Een klein voorbeeld:

```
VAR
   RIJ: Array[1..10] of Byte;
```

Hier wordt een variabele RIJ gedeclareerd van het type Array of Byte. De lengte van de array is 10 elementen. Met deze declaratie worden er in het computergeheugen 10 bytes gereserveerd voor de variabele RIJ. De verschillende elementen zijn bereikbaar met behulp van de index die tussen vierkante haken wordt opgegeven. In dit geval loopt de index van 1 tot en met 10. Met RIJ[5] bereik je dan het vijfde element van de array, RIJ[9] geeft je de inhoud van het negende element. Je kunt je dit array als volgt voorstellen:

```
((((((((((((((((((((((((((((((((((((((((
(   (   (   (   (   (   (   (   (   (   (
((((((((((((((((((((((((((((((((((((((((
  1   2   3   4   5   6   7   8   9   10
```

Nog een voorbeeld:

```
VAR
   RIJ: Array[101..200] of Byte;
```

Deze declaratie reserveert 100 bytes in het geheugen, alleen loopt de index nu van 101 tot en met 200. Je kunt je dit als volgt voorstellen:

```
((((((((((( (( (( ((((((((((( 
(   (   (               (   (   (
((((((((((( (( (( ((((((((((( 
 101          .....          200
```

Deze array's zijn beide één-dimensionaal. Dat wil zeggen dat het enkelvoudige rijen van elementen zijn.

Het is ook mogelijk een twee-dimensionale array te declareren:

```
VAR
   RIJ: Array[1..100,1..100] of Byte;
```

Hier worden 100 arrays gereserveerd die alle 100 bytes bevatten. Het vijfde element uit de tiende array wordt bereikt met:

```
RIJ[10,5]
```

In gedachten kun je deze twee-dimensionale array visualiseren als een velletje ruitjespapier met honderd bij honderd vakjes.

Als je nog een komma achter de declaratie van dit array zou plaatsen, met opnieuw een index die loopt van [1..100], dan zou je een drie-dimensionale array krijgen. Hierbij kun je je een kubus voorstellen.

Wees buitengewoon voorzichtig met het gebruik van dit soort meer-dimensionale arrays. De één-dimensionale array uit het voorbeeld gebruikt honderd bytes. De twee-dimensionale array gebruikt 100 x 100 = 10.000 bytes. De drie-dimensionale array gebruikt 100 x 100 x 100 bytes. Dit is al een miljoen bytes. Je ziet het: meer-dimensionale array's kosten heel wat geheugenruimte. Declareer daarom nooit teveel, maar precies zoveel als je nodig denkt te hebben om de gegevens van hetzelfde type in op te slaan.

In het volgende programma gaan we werken met een array, en ook met een WHILE DO-lus:

```
PROGRAM TYPE_3;
USES CRT;

VAR
   I  : Byte;
   RIJ: Array[1..100] of Byte;

BEGIN
   ClrScr;
   I := 1;
   WHILE I <= 100 DO
   BEGIN
      RIJ[I] := I;
      Inc(I)
   END;
   I := 1;
   WHILE I <= 100 DO
   BEGIN
      Write(RIJ[I],' ');
      IF I MOD 20 = 0 THEN Writeln;
      Inc(I)
   END;
   Writeln('De laagste waarde = ',Low(RIJ),
          ' De hoogste waarde = ',High(RIJ));
   Readln
END.
```

Regels: **Omschrijving**:

[1] 3-5 Declareer de benodigde variabelen.
7 Veeg het scherm schoon.
[2] 8-13 Zet zolang I kleiner of gelijk is aan 100 de waarde van I in het bijbehorende element.
[3]14-20 Zet de inhoud van de array op het scherm en maak hier regels van.
[4] 21-22 Zet de hoogste en de laagste waarde in RIJ op het scherm.

Toelichting:

[1]De variabele I van het type Byte zullen we straks nodig hebben om een index mee aan te geven. De variabele RIJ is een een-dimensionale array met 100 elementen.

[2]Herhaling is erg belangrijk bij het programmeren. Free Pascal geeft ons een aantal mogelijkheden om herhalingen uit te voeren. Zo'n herhaling wordt een lus genoemd. Ook wordt vaak het Engelse woord "loop" gebruikt. In het programma TYPE_3 zien we een voorbeeld van de WHILE DO-lus. Direct na de vergelijking WHILE I <= 100 DO staat een BEGIN. Daarachter staan de opdrachten die op regel 13 door een END worden afgesloten. Alles tussen deze BEGIN en END behoort tot de lus.

Op regel 8 krijgt I de waarde 1. Op regel 9 wordt met behulp van de operator <= gekeken of I kleiner of gelijk aan 100 is (zie voor de werking van operatoren hoofdstuk 13). Als dit het geval is, wordt begonnen met de uitvoering van de lus. In het voorbeeld wordt hieraan voldaan, want I is immers net op 1 gezet. Op regel 11 wordt vervolgens de waarde van I in RIJ[I] geplaatst. De eerste keer betekent dit dus dat in het eerste element van de array (RIJ[I]) de waarde 1 staat.

Op regel 12 wordt I met 1 verhoogd. Hiervoor gebruiken we de Free Pascal-procedure Inc. Inc(I) heeft hetzelfde resultaat als I := I + 1. Als er gestaan zou hebben Inc(I,10), zou I met 10 verhoogd zijn. Als je niet wilt optellen maar aftrekken, gebruik je de Free Pascal-procedure Dec. I heeft nu de waarde 2 gekregen. In de vergelijking wordt gekeken of I nog steeds kleiner of gelijk aan honderd is. Als dit het geval is, wordt de lus opnieuw uitgevoerd. Dit proces gaat net zolang door tot I de waarde 101 heeft. Als dit geconstateerd wordt, dan voldoet I niet meer aan de voorwaarde dat hij kleiner of gelijk aan 100 moet zijn en vervolgt het programma met de opdracht op regel 14, waar I weer gelijk aan 1 gemaakt wordt.

[3]Om terug te kunnen lezen wat we zojuist in de variabele RIJ gezet hebben, wordt er opnieuw een WHILE-lus ingegaan. Zoals je ziet heeft de lus weer dezelfde vorm. Alleen de opdrachten tussen BEGIN en END zijn anders. In dit geval wordt de waarde die in RIJ[I] staat op het scherm afgedrukt, gevolgd door een spatie. Op regel 18 is weer iets bijzonders aan de hand. Hier worden we geconfronteerd met een IF THEN-constructie. Deze vergelijkingsopdracht is bestemd om condities in de vorm van: "Als hieraan voldaan wordt doe dan iets" te testen. In dit geval wordt gekeken of de rest van de deling I:20 gelijk aan 0 is. Hiervoor gebruiken we de operator MOD. Met de operator MOD kun je de restwaarde van een deling berekenen, zoals: 5 MOD 2 = rest 1. Als de uitkomst I MOD 20 het getal 0 is, zijn er dus 20 elementen, of een veelvoud daarvan, op het scherm afgedrukt. In dat geval willen we naar een nieuwe regel springen. Dit doen we met een Writeln-opdracht.

Het verschil tussen Write en Writeln is dat bij Write de cursor achter de afgebeelde tekst blijft staan, terwijl bij Writeln de cursor naar het begin van de volgende regel wordt gebracht; je geeft dus een soort Enter. Als er vervolgens een nieuwe Write- of Writeln-opdracht volgt, dan wordt die aan het begin van de nieuwe

regel gezet. Tekst wordt nu eenmaal afgebeeld vanaf de plaats waar de cursor staat.

[4]In de laatste regels worden met de Free Pascal-functies Low en High de laagste en de hoogste waarde opgezocht in de Array RIJ. De typen en waarden die door High en Low geretourneerd worden, hangen af van het type parameter dat naar ze toegezonden wordt. Bij een array wordt de laagste en de hoogste waarde geretourneerd.

Als je een variabele van een opsommingstype, zoals een integer (zie paragraaf 4.6), naar Low stuurt, dan wordt de laagst mogelijke waarde van een integer geretourneerd (-32768). High retourneert de hoogst mogelijke waarde (+32767). Stuur je een tekst naar Low, dan krijg je een 0 terug. Bij het sturen van een tekst naar High, wordt de gedeclareerde afmeting geretourneerd.

Als we het programma uitvoeren, krijgen we een scherm te zien met de getallen 1 tot en met 100, verdeeld over vijf regels. Onder deze vijf regels wordt de hoogste en de laagste waarde in RIJ op het scherm gezet:

*Fout! Bladwijzer niet gedefinieerd.**Afbeelding 5***

4.5 Het type String

Nu we weten wat een type Char is en hoe een array werkt, moet het type String een fluitje van een cent zijn. Strings worden gebruikt om woorden of zinnen in op te bergen. Een string is een rij van letters, of zoals we dat behandeld hebben: een Array of Char.

Bij de declaratie van een string kun je opgeven hoe lang de string maximaal wordt. Een declaratie kan zijn:

```
WOORD: String[25];
```

Deze declaratie reserveert voldoende geheugen om een woord van 25 letters (inclusief spaties) in op te bergen. De compiler reserveert 26 bytes voor een string waarin maximaal 25 letters kunnen staan. Die extra byte, de lengtebyte genaamd, is nodig omdat Free Pascal in de eerste byte van de string bijhoudt hoeveel letters er feitelijk in de string staan.

Als we een string met een lengte van 25 bytes declareren, wil dat nog niet zeggen dat het woord dat wij daarin willen plaatsen ook 25 letters groot is. Het geeft alleen aan hoeveel geheugenruimte we willen reserveren. In de eerste byte wordt automatisch bijgehouden hoe groot de feitelijke lengte van het woord is. De declaratie:

```
WOORD: String[25];
```

is gelijk aan de volgende:

```
WOORD: Array [0..25] of Char;
```

In WOORD[0] staat dan hoe lang het woord feitelijk is. Als we een string binnen een programma als een array zouden declareren, dan zouden we de lengtebyte zelf bij moeten houden. Door niet een array maar een string te declareren, wordt de lengte automatisch bijgehouden.

Een string kan maximaal 255 letters lang zijn (een spatie telt ook voor een letter). Dit is eigenlijk wel logisch, want de maximale waarde die in één byte gezet kan worden is ook 255. Als we op dezelfde manier langere strings zouden willen maken, dan zouden we meer dan één lengtebyte nodig hebben.

Als een string gedeclareerd wordt zonder vierkante haken en lengte-aanduiding, dan wordt door de compiler aangenomen dat je een string van de maximale lengte wilt hebben en worden er 256 bytes in het geheugen gereserveerd.

Free Pascal heeft een groot aantal mogelijkheden om strings te bewerken. In het volgende programma TYPE_4 laat ik zien welke dat zijn:

```
   PROGRAM TYPE_4;
   USES CRT;

   VAR
ZIN: String;

   BEGIN
      ClrScr;
      ZIN := 'Pascal leren is bepaald niet moeilijk';
      Writeln(ZIN);
      Writeln('De variabele ZIN is ',Length(ZIN),
              ' lettertekens groot');
      Writeln('In het lengtebyte staat: ',ZIN[0],
              ' Dat is nummer ',Length(ZIN),
              ' uit de ASCII-tabel');
      Writeln('niet staat op de ',Pos('niet',ZIN),
              'e positie binnen de string');
      Delete(ZIN,25,5);
      Writeln('Na verwijdering van niet staat er: ',ZIN);
      Writeln('Nu staat op de 25e positie: ',
         Copy(ZIN,25,8));
      Insert('in het begin ',ZIN,25);
      Writeln(ZIN);
      ZIN := Concat(ZIN,' maar toch ook wel leuk');
      Writeln(ZIN);
      Writeln('In het lengtebyte staat: ',ZIN[0],
              ' Dat is nummer ',Length(ZIN),
              ' uit de ASCII-tabel');
      Readln
   END.
```

Na uitvoering van dit programma krijgen we een overzicht van wat we allemaal met de inhoud van ZIN hebben gedaan, zoals afbeelding 6 laat zien.

*Fout! Bladwijzer niet gedefinieerd.**Afbeelding 6***

Fout! Bladwijzer niet gedefinieerd.**Regels:** **Omschrijving:**

[1]3-4 Declareer de variabele ZIN.
[2] 7-8 Geef ZIN een beginwaarde en zet deze waarde op het scherm.
[3]9-10 Zet het aantal lettertekens van ZIN op het scherm.
[4]11-13 Laat zien wat er in de lengtebyte staat.
[5]14-15 Geef aan op welke positie het woordje "niet" staat.
[6]16-17Verwijder het woordje "niet" uit ZIN en laat zien hoe ZIN er nu uitziet.
[7]18-19 Laat zien welk woord er nu op de vijfentwintigste positie staat.
[8]20-21 Voeg op de vijfentwintigste positie van ZIN in: "in het begin " en laat zien wat er nu in ZIN staat.
[9]22-23 Voeg ZIN samen met : " maar toch ook wel leuk" en laat de nieuwe ZIN zien.
[10]24-26 Laat de inhoud van de lengtebyte zien.

Fout! Bladwijzer niet gedefinieerd.**Toelichting:**

[1]De variabele ZIN wordt hier gedeclareerd zonder lengte-aanduiding tussen vierkante haken en heeft een maximale lengte. Dit heeft tot gevolg dat in het geheugen 256 bytes gereserveerd worden voor de variabele ZIN en dat er maximaal 255 tekens in opgeslagen kunnen worden.

[2]Een string krijgt een waarde toegewezen door middel van de toewijzingsoperator, gevolgd door een tekst tussen aanhalingstekens.

[3]De Free Pascal-functie Length geeft het aantal letters in de string die tussen de haakjes geplaatst wordt. In dit geval geeft Length(ZIN) de waarde 37 terug. Denk erom dat voor de berekening van het aantal letters een spatie ook als letter wordt meegeteld.

[4]Vervolgens laten we in de Write-opdracht zien wat er in de lengtebyte (het eerste byte) van de string staat. Een string is dus eigenlijk een array van letters. In dit geval is ZIN dus een Array [0..255] of Char. Dat betekent dat ZIN[0] de lengtebyte is. Door ZIN[0] af te laten drukken, kunnen we de inhoud ervan zien. Als je de ASCII-tabel uit bijlage 7 erbij neemt, kun je zien dat de lengte, opgevraagd met de functie Length, overeenstemt met het nummer van de afgedrukte letter.

[5]De Free Pascal-functie Pos zoekt in een string naar een opgegeven substring. Een substring is een deel van een string. In het woord "huismus" bijvoorbeeld zijn "huis" en "mus" substrings. Als de substring gevonden wordt, retourneert Pos de beginpositie van de substring in de string. Tussen de haakjes wordt eerst de substring tussen aanhalingstekens opgegeven, gevolgd door een komma en de naam van de string waarin gezocht moet worden. Als de substring niet gevonden wordt, retourneert Pos een 0.

[6]We weten nu op welke positie het woordje "niet" begint. We verwijderen "niet" uit de string met de Free Pascal-procedure Delete. Tussen de haakjes zetten we eerst de naam van de variabele en vervolgens de positie waar de te verwijderen substring begint. Daarna wordt het aantal te verwijderen bytes opgegeven. De opdracht:

```
Delete(ZIN,25,5);
```

verwijdert 5 bytes uit de variabele ZIN, te beginnen op positie 25. Het woordje "niet" en de bijbehorende spatie zijn nu uit de tekst verdwenen.

[7]Zoals het mogelijk is een substring uit een string te verwijderen, zo is het ook mogelijk om een substring uit een string te kopiëren. Hiervoor wordt de procedure Copy gebruikt. Deze procedure werkt op dezelfde wijze als Delete. De opdracht:

```
Copy(ZIN,25,8);
```

levert 8 bytes vanaf positie 25 als substring op. Omdat het woordje "niet" verwijderd is, begint nu het woord "moeilijk" op de vijfentwintigste positie.

[8] Als verwijderen en kopiëren mogelijk is, dan zal invoegen van een substring ook wel mogelijk zijn. En inderdaad, daarvoor hebben we Insert. Deze procedure werkt net even anders dan de twee voorgaanden. De opdracht:

```
Insert('in het begin ',ZIN,25);
```

voegt de substring "in het begin " op de vijfentwintigste positie in tussen de reeds bestaande inhoud van ZIN.

[9]Tenslotte moet je ook nog twee strings samen kunnen voegen. De opdracht:

```
ZIN := Concat(ZIN,' maar toch wel leuk');
```

plakt de substring: " maar toch wel leuk" aan de variabele ZIN vast. Concat is een afkorting van het Engelse woord "concatenate", dat "samenvoegen" betekent. Voor Concat bestaat ook nog een eenvoudiger schrijfwijze met hetzelfde effect:

```
ZIN := ZIN + ' maar toch wel leuk';
```

[10]Hier wordt net als in regel 11 tot en met 13, de inhoud van de lengtebyte getoond.

4.6 Integertypen

Integertypen zijn bestemd om gehele getallen in op te slaan. Het begrip geheel getal staat hier tegenover een gebroken getal. In een gebroken getal staan de gehele getallen vóór de decimale punt en staat het breukgedeelte daarachter. Een integer-type kent alleen gehele getallen. Oorspronkelijk kende Pascal alleen het type "integer", waarvan niet gespecificeerd was hoe groot de getallen waren die je er in op kon slaan. Zoiets programmeert vervelend en daarom voorzien de meeste compilers in meerdere integertypen, waarbij iedere verschillende eigenschappen heeft. Zo ook Free Pascal. Er zijn twee soorten integertypen, namelijk met voorteken en zonder voorteken. Integertypen zonder voorteken kunnen alleen positieve gehele getallen bevatten.

Free Pascal kent de volgende integertypen zonder voorteken:

```
|Naam:                |Bereik:                          |Aantal bytes:  |
|byte                 |0..255                           |1              |
|word                 |0..65535                         |2              |
|cardinal of longword |0..4294967296                    |4              |
|qword                |0..1844674407370955 1615         |8              |
```

Binnen Free Pascal neemt het type "word" twee bytes in beslag. Toen de IBM-PC ontworpen werd was dit nog een 16-bits computer, en was een machinewoord, dat is met hoeveel bits de microprocessor in één keer bewerkingen kan uitvoeren, daarom 16-bits breed, oftewel vormen twee bytes samen een woord. Sindsdien is binnen de PC de definitie van een woord nog nooit uitgebreid en daarom noemen we een woord vandaag nog steeds twee bytes, ook al werkt de processor eigenlijk met 32- of 64-bits woorden.

In twee bytes kun je 65536 verschillende waarden opslaan. Met de 0 meegenomen levert dat op dat je de getallen 0 tot en met 65535 in een word kunt opslaan. Maar hoe zit het als we ook negatieve getallen willen opslaan, oftewel een integer met voorteken? We kunnen dan nog steeds 65536 verschillende getallen opslaan in het integertype, alleen we moeten het bereik wat verleggen. Het integertype smallint is het hoogste gehele getal dat in de integer past +32767 is, en het laagste getal -32768.

Free Pascal kent de volgende integertypen met voorteken:

```
|Naam:                |Bereik:                          |Aantal bytes:  |
|shortint             |-128..127                        |1              |
|smallint             |-32768..32767                    |2              |
|longint              |-2147483648.. 2147483647         |4              |
|int64                |-9223372036854775808..           |8              |
|                     |9223372036854775807              |               |
```

Zoals we boven schreven, war er oorspronkelijk in Pascal alleen maar een type met de naam "integer". Omdat niet gespecificeerd was hoe groot dat type moest zijn, verschilde dat nogal eens in de praktijk. Free Pascal kent ook een "integer", maar, de grootte is afhankelijk van de compilermode. Als je de compiler in Turbo Pascal- of FPC-mode zet, dan is een "integer" hetzelfde als een "smallint". Als je de compiler in Objfpc-, Delphi- of Macpas-mode zet, dan is een "integer" hetzelfde als een longint.

De maximale waarde die in een integer gezet kan worden, wordt weergegeven door de binnen Free Pascal voorgedefinieerde constante MAXINT. Deze constante heeft een waarde van +32767. In paragraaf 4.13 komt het werken met constanten aan de orde.

Om te bepalen of het om een negatief of een positief getal gaat, wordt bij integertypen met voorteken de

hoogste bit als voortekenbit gebruikt. Als die bit op 1 staat, gaat het om een negatief getal. Staat deze op 0, dan is het getal positief. De twee bytes van een type Smallint omvatten zoals gezegd 16 bits. Deze bits zijn van rechts naar links genummerd als de bits 0 tot en met 15:

```
[15] [14] [13] [12] [11] [10] [9] [8] [7] [6] [5] [4] [3] [2] [1] [0]
```

Bit 15 is de hoogste bit en wordt dus als voortekenbit gebruikt. Als bit 15 op 1 staat, dan gaat het om een negatief getal. Staat de bit op 0, dan hebben we met een positief getal te maken.

In het volgende programma worden integertypen gebruikt:

```
PROGRAM TYPE_5;
USES CRT;

VAR
   GEWOON: Smallint;
   KLEIN : Shortint;
   GROOT : Longint;
   WOORD : Word;

BEGIN
   GEWOON := -23415;
   KLEIN  := GEWOON;
   GROOT  := GEWOON;
   WOORD  := GEWOON;
   ClrScr;
   Writeln('KLEIN = ',KLEIN);
   Writeln('GROOT = ',GROOT);
   Writeln('WOORD = ',WOORD);
   Readln
END.
```

Toelichting:

In het programma TYPE_5 worden vier variabelen gedeclareerd van respectievelijk het type Integer, Shortint, Longint en Word. Aan het begin van de uitvoering van het programma krijgt de variabele GEWOON de waarde -23415. Dit is duidelijk een waarde die groter is dan de waarde die in de variabele KLEIN past. Deze laatste is van het type Shortint met een waardebereik van +127 tot en met -128. De andere twee zijn groot genoeg. GROOT heeft vier bytes ter beschikking en WOORD heeft er twee. Deze laatste is echter weer niet geschikt om negatieve getallen in op te slaan.

Vervolgens moet de inhoud van KLEIN, GROOT en WOORD op het scherm gezet worden. Nu blijkt dat GROOT de juiste waarde weergeeft, maar dat de inhoud van KLEIN en WOORD niet de waarde weergeven die ze toegekend hebben gekregen. Zij geven respectievelijk de waarden -119 en 42121 terug.

We kunnen dus concluderen dat de inhoud van de verschillende typen onderling goed uitwisselbaar is, maar dat dit alleen lukt als er een waarde overgedragen wordt die in het betreffende type past. Is dit niet het geval, dan wordt de betreffende variabele van een verkeerde waarde voorzien. Het is de verantwoordelijkheid van de programmeur om ervoor te zorgen dat deze situatie zich niet voordoet. Dergelijke fouten kunnen aanleiding zijn tot wanhopige zoekacties.

Het programma TYPE_6 laat het gebruik van de rekenkundige operatoren met integers zien:

```
PROGRAM TYPE_6;
USES CRT;

VAR
   GETAL, UITKOMST: Integer;


BEGIN
   ClrScr;
   GETAL := 67;
   UITKOMST := GETAL + 10;
   Writeln(GETAL, ' plus 10 = ',UITKOMST);
   UITKOMST := GETAL - 10;
   Writeln(GETAL,' min 10 = ',UITKOMST);
   UITKOMST := GETAL * 10;
   Writeln(GETAL,' maal 10 = ',UITKOMST);
   UITKOMST := GETAL DIV 10;
   Writeln(GETAL,' gedeeld door 10 = ',UITKOMST);
   UITKOMST := GETAL MOD 10;
   Writeln('De rest van ',GETAL,
           ' gedeeld door 10 = ',UITKOMST);
   Readln;
END.
```

Als je het programma draait, zul je het verschil tussen de operatoren DIV en MOD zien (zie ook hoofdstuk 13). Je moet er rekening mee houden dat DIV en MOD slechts bruikbaar zijn bij integers en niet bij het volgende te bespreken type Real.

4.7 Het type Real

Het type Real is bestemd voor de opslag van gebroken getallen en neemt 6 bytes in het geheugen in beslag. Een gebroken getal heeft achter de decimale punt een breukgedeelte staan. Het type Real werkt nauwkeurig op getallen van elf cijfers of minder. Het volgende getal wordt nauwkeurig weergegeven:

```
1234567.1234
```

Bij het getal:

```
1234567.1234567
```

worden vanaf het twaalfde cijfer nullen toegevoegd. Dit getal wordt dus als volgt weergegeven:

```
1234567.1234000
```

Net als bij integers gebruiken we operatoren en functies om getallen van het type Real te manipuleren. Het programma TYPE_7 laat dit zien:

```
PROGRAM TYPE_7;
USES CRT;

VAR
   GETAL, UITKOMST: Real;

BEGIN
   ClrScr;
   GETAL := 67.575;
   UITKOMST := GETAL + 10;
   Writeln(GETAL:2:3, ' plus 10 = ',UITKOMST:2:3);
   UITKOMST := GETAL - 10;
   Writeln(GETAL:2:3,' min 10 = ',UITKOMST:2:3);
   UITKOMST := GETAL * 10;
   Writeln(GETAL:2:3,' maal 10 = ',UITKOMST:3:3);
   UITKOMST := GETAL / 10;
   Writeln(GETAL:2:3,' gedeeld door 10 = ',
           UITKOMST:1:4);
   Writeln('Het integere deel van ',GETAl:2:3,' is ',
            Int(GETAL):2:3);
   Writeln('Het breukgedeelte van ',GETAL:2:3,' is ',
            Frac(GETAL):2:3);
   Writeln('Afgerond is ',GETAL:2:3,' gelijk aan ',
            Round(GETAL));
   Writeln('Afgesneden is ',GETAL:2:3,' gelijk aan ',
            Trunc(GETAL));
   Readln
END.
```

Toelichting:

Over bovenstaand programma is wel iets te vertellen. In de eerste plaats zien we dat in de Writeln-opdrachten achter de variabele GETAL dubbele punten met cijfers staan. Dit bepaalt de opmaak van het getal. De opdracht:

```
Writeln(GETAL:2:3...);
```

wil zeggen: druk GETAL af met 2 plaatsen vóór en 3 plaatsen achter de decimale punt. Als we deze opmaak achterwege zouden laten, dan zou het getal afgedrukt worden in de wetenschappelijke notatie:

```
6.7575000000E+01
```

Met deze notatie hebben veel mensen moeite. Toch is het niet zo moeilijk te "ontcijferen". Het getal bestaat uit een mantisse en een exponent. "6.7575000000" is de mantisse. "E+01" is de exponent. In de exponent staat het aantal machten van 10 dat gebruikt moet worden om het "echte" getal te berekenen. Bij E+01 schuift de decimale punt één plaats naar rechts op. De uitkomst is dan 67.575. Dit is de waarde die we in GETAL gezet hebben. Zou er E+02 staan, dan zou de punt twee plaatsen naar rechts opschuiven en zou de uitkomst 675.75 zijn. Als er E-01 zou staan, dan zou de punt één plaats naar links opschuiven en zou de uitkomst 0.67575 zijn.

De rekenkundige operatoren worden op dezelfde wijze gebruikt als bij de integers, met uitzondering van DIV en MOD. De bewerking MOD is niet mogelijk met getallen van het type Real en in plaats van DIV wordt het vooroverliggende streepje "/" gebruikt als operator om te delen.

Een gebroken getal bestaat dus uit een geheel getal vóór de decimale punt en een breukgedeelte achter deze punt. Het deel vóór de punt wordt ook wel het "integere deel" genoemd en het breukgedeelte de "fractie". Het is mogelijk om het integere deel en de breuk van het getal van elkaar te scheiden. Hiervoor gebruiken we de Free Pascal-procedures Int en Frac. Als je de uitkomst hiervan aan een andere variabele wilt geven, moet die variabele van het type Real zijn.

Round rondt een getal met een breukgedeelte af. Lager dan 0.5 wordt afgerond naar beneden, en 0.5 en hoger wordt afgerond naar boven. Trunc snijdt eenvoudig het integere deel van een getal af. De uitkomst van Round en Trunc kan meegedeeld worden aan een variabele van zowel het type Integer als van het type Real.

4.8 Type-conversie

Soms is het nodig om van een getal een string te maken en soms moet van een string een getal gemaakt worden. Een voorbeeld waarbij het noodzakelijk is om numerieke waarden om te zetten naar een string is de datum. Deze bestaat uit een drietal cijfercombinaties, van elkaar gescheiden door een streepje. Omdat je een datum niet als een getal kunt weergeven, maak je van de verschillende cijfercombinaties een string en plak je deze aan elkaar met streepjes ertussen. Andersom kan het ook nodig zijn om met een deel van de datum een rekenkundige bewerking uit te voeren. Het deel van de datum dat je nodig hebt moet dan, om er mee te kunnen rekenen, omgezet worden naar een numerieke waarde. Dit omzetten noemen we converteren. De functies Str en Val voeren deze conversies uit. Het programma TYPE_8 laat dit zien:

```
PROGRAM TYPE_8;
USES CRT;

VAR
   REAL_GETAL  : Real;
   INT_GETAL   : Integer;
   STRING_GETAL: String;
   CODE        : Integer;

BEGIN
   ClrScr;
   REAL_GETAL := 67.575;
   Str(REAL_GETAL:2:3,STRING_GETAL);
   Writeln('In STRING_GETAL staat nu: ',STRING_GETAL);
   INT_GETAL := Round(REAL_GETAL);
   Str(INT_GETAL,STRING_GETAL);
   Writeln('In STRING_GETAL staat nu: ',STRING_GETAL);
   STRING_GETAL := '13.345';
   Val(STRING_GETAL,REAL_GETAL,CODE);
   Writeln('Nu staat in REAL_GETAL: ',
            REAL_GETAL:2:3);
   STRING_GETAL := '13';
   Val(STRING_GETAL,INT_GETAL,CODE);
   Writeln('Nu staat in INT_GETAL: ',INT_GETAL);
   STRING_GETAL := '12G5';
   Val(STRING_GETAL,INT_GETAL,CODE);
   Writeln('In INT_GETAL staat: ',INT_GETAL,
        ' In CODE staat: ',CODE);
     Readln
END.
```

Regels: **Omschrijving:**

[1]3-7 Declareer de noodzakelijke variabelen.
[2]11-12Converteer de real naar een string en laat zien welke waarde nu in de string staat.
[3]13-15Rond de waarde van de real af en zet die in de integer. Converteer vervolgens de integer naar een string en laat zien welke waarde nu in de string staat.
[4]16-19 Zet een waarde in de string en converteer deze naar een real. Laat zien wat er in de real staat.
[5]20-22 Zet een waarde in de string en converteer deze naar een integer. Laat zien wat er in de integer staat.
[6]23-26Zet een foutieve waarde in de string en converteer deze naar een integer. Laat zien wat er in de integer en wat er in CODE staat.

Toelichting:

[1]Om een goed inzicht te krijgen in de werking van de conversie, hebben we een variabele van het type Real, van het type Integer en van het type String nodig. Om de conversie met Val uit te kunnen voeren, hebben we bovendien een integer nodig waarin aangegeven wordt of de operatie gelukt is of niet.

[2]Als we een variabele van het type Real naar het type String willen converteren, moeten we de Free Pascal-functie Str gebruiken. Als we de string niet in wetenschappelijke notatie willen hebben, moeten we een formaat aan de real meegeven. Dit gaat op dezelfde manier als het opgeven van een formaat bij Write en Writeln, namelijk door een dubbele punt en een cijfer op te geven.

[3]Hier gebruiken we Round om de afgeronde waarde die in REAL_GETAL staat, in INT_GETAL te zetten. INT_GETAL wordt dus op 68 gezet. Bij een integer hoeven we geen formaat op te geven. De string wordt dan zo lang als noodzakelijk is om het getal op te bergen. Geven we wel een formaat op als:

```
Str(INT_GETAL:4,STRING_GETAL);
```

dan wordt er een string gemaakt van 4 bytes en wordt het geconverteerde getal daar van rechts naar links in gezet. De onbenutte ruimte wordt opgevuld met spaties.

[4] Nu gaat de zaak andersom lopen. We zetten een gebroken getal in een string en we gebruiken Val om de string te converteren naar een real. Als na de uitvoering van de conversie de variabele CODE op 0 staat, dan is de conversie succesvol verlopen.

[5]De conversie van een integer gaat op dezelfde manier als de conversie van een real.

[6]Hier wordt bewust een fout gemaakt. 12G5 is natuurlijk geen getal. Voor een string maakt het niets uit. Een string is immers slechts een rij van lettertekens. Als we de string met de waarde 12G5 proberen te converteren naar een numerieke waarde, dan zal dit mislukken. Op dat moment krijgt CODE een waarde die ongelijk is aan 0. In dit geval zal CODE de waarde 3 hebben, omdat de derde letter in de string niet naar een cijfer geconverteerd kan worden. Een cijfer G bestaat nu eenmaal niet.

4.9 Rangorde-typen

Integer- en Real-typen zijn rangorde-typen: zij hebben een minimaal en een maximaal waardebereik. In het Engelse jargon worden dit "ordinal types" genoemd. In Free Pascal kunnen we ook zelf dergelijke typen definiëren. Deze typen worden gemaakt als een opsomming van een geordende reeks waarden (enumerated

types). Het volgende programma laat zien hoe dat in zijn werk gaat:

```
PROGRAM TYPE_9;
USES CRT;

TYPE
   Kleuren =(Zwart, Blauw, Groen, Cyaan, Rood,
             Magenta, Bruin, Grijs);

BEGIN
   TextBackground(Ord(Blauw));
   TextColor(Ord(Grijs));
   ClrScr;
   GotoXY(15,10);
   Write('Zo kun je ook met kleuren werken');
   Readln
END.
```

Toelichting:

Een variabele van het type Kleuren zal alleen de aanduidingen accepteren die bij de type-declaratie zijn opgegeven, dus Zwart, Blauw, Groen, enzovoort. De opgegeven kleuren zijn tegelijkertijd in rangorde geplaatst. Zij hebben een nummer gekregen dat afgeleid is van de plaats die ze bij de type-definitie innemen. Zwart is gelijk aan 0, Blauw is gelijk aan 1, Groen heeft de waarde 2, enzovoort. De Free Pascal-functie Ord geeft de rangorde terug.

De opdracht:

```
TextBackground(Ord(Blauw));
```

heeft hetzelfde effect als de opdracht:

```
TextBackground(1);
```

4.10 Het type Boolean

Het type Boolean is genoemd naar de 19e-eeuwse Engelse wiskundige George Boole. Deze meneer Boole was een van de eersten die de logica wiskundig benaderde. Hij heeft een algebra ontwikkeld die gebaseerd is op "waar" of "niet waar". In deze eeuw van computers komt het wetenschappelijke werk van George Boole goed van pas. Als we "niet waar" vertalen in 0 en "waar" vertalen in 1, ligt dit uitgangspunt precies in lijn met de mogelijkheden van computers.

In Free Pascal heeft het Booleaanse type een voorgedefinieerde rangorde in de volgende vorm:

```
TYPE
         Boolean = (False, True);
```

Variabelen van dit type kunnen dus slechts twee waarden aannemen: False of True, ofwel "niet waar" of "waar". In het programma TYPE_10 kunnen we zien waarvoor een Boolean-variabele bestemd is en hoe deze gebruikt wordt:

```
PROGRAM TYPE_10;
USES CRT;

VAR
    UITKOMST: Boolean;

BEGIN
   ClrScr;
   UITKOMST := 5 * 2 = 10;
   Writeln('5 maal 2 = 10. Dit is ',UITKOMST);
   UITKOMST := 5 * 2 = 11;
   Writeln('5 maal 2 = 11. Dit is ',UITKOMST);
   UITKOMST := 10 < 100;
   Writeln('tien is kleiner dan honderd. Dit is ',
            UITKOMST);
   UITKOMST := 10 > 100;
   Writeln('tien is groter dan honderd. Dit is ',
            UITKOMST);
   Readln
END.
```

Toelichting:

Het Booleaanse type wordt gebruikt om een bewering als "waar" of "niet waar" te kwalificeren. Zo'n bewering wordt ook wel een expressie genoemd. De expressie:

```
5 * 2 = 10
```

vertelt dat de uitkomst van 5 maal 2 het getal 10 is. Dit is een correcte bewering. De vergelijking geeft een True ("waar") terug. In de Boolean-variabele UITKOMST wordt dus de waarde 1 gezet.

De volgende expressie is onjuist:

```
5 * 2 = 11
```

Daarom zal de uitkomst van deze expressie een 0 (False) zijn.

4.11 Het type Subrange

Je kunt in Free Pascal een type maken dat slechts een beperkt bereik heeft. De Engelse term hiervoor is "subrange type". Als variabelen van een type met een beperkt bereik een waarde toegewezen krijgen die buiten hun bereik ligt, dan onderbreekt de compiler de uitvoering van het programma met foutmelding 76. Het programma TYPE_11 laat dit zien:

```
PROGRAM TYPE_11;
USES CRT;
TYPE
   Letters = 'A'..'Z';
VAR
   LETTER: Letters;

BEGIN
   ClrScr;
   LETTER := 'V';
   Writeln(LETTER);
   LETTER := 'v';
   Writeln(LETTER);
   Readln
END.
```

Toelichting:

Met de volgende toewijzing heeft de compiler geen moeite:

```
LETTER := 'V';
```

De hoofdletter "V" ligt immers binnen het bereik 'A'..'Z'. De toewijzing:

```
LETTER := 'v';
```

wordt door de compiler echter niet geaccepteerd. De kleine letter "v" bevindt zich buiten het bereik van 'A'..'Z'. Als we in de ASCII-tabel kijken, zien we dat de letters 'A'..'Z' de rangorde 65 tot en met 90 hebben en dat de kleine letter "v" nummer 118 heeft. Hieruit volgt dat "v" buiten het bereik van 'A'..'Z' ligt.

De zaak ligt even anders als LETTERS zijn waarde meegedeeld zou krijgen door een andere variabele. Het programma wordt geaccepteerd, omdat de compiler bij het compileren niet kan weten welke waarde er in een variabele staat.

Het onderstaande programma demonstreert dit:

```
PROGRAM TYPE_12;
USES CRT;

TYPE
   Letters = 'A'..'Z';

VAR
   LETTER  : Letters;
   LETTER_2: Char;

BEGIN
   ClrScr;
   LETTER := 'V';
   Writeln(LETTER);
   LETTER_2 := 'v';
   LETTER := LETTER_2;
   Writeln(LETTER);
   Readln
END.
```

Toelichting:

In het programma TYPE_12 is LETTER_2 gedeclareerd als variabele van het type Char. Deze variabele krijgt de waarde "v". Vervolgens wordt de waarde van LETTER_2 in LETTER gezet.

Of het programma onderbroken wordt tijdens de uitvoering, is afhankelijk van het feit of in de IDE (onder «*Options*» en «*Compiler*») de optie «*Range checking*» "aan" of "uit" staat. Staat deze optie aan, dan wordt de uitvoering onderbroken door foutmelding 201: "range check error". Dat wil zeggen dat de bereikcontrole een fout meldt. Staat de optie niet aan, dan accepteert LETTER rustig de kleine letter "v" die in LETTER_2 staat. Dat kon echter nooit de bedoeling zijn, want je definieert nu eenmaal een subrange-type om een aantal waarden uit te sluiten. Zet dus de optie «*Range checking*» standaard op "aan".

4.12 Het type Set

Een set is een verzameling waarden van een bepaald type. Je kunt bijvoorbeeld de waarden "A" tot en met "Z" definiëren als de set [’A’..’Z’]. Het type Set wordt gebruikt om te controleren of een voorkomende waarde zich binnen het waardebereik van de set bevindt.

Laten we eens aannemen dat een bepaalde vraag op het scherm met "Ja" of "Nee" beantwoord moet worden. Je zou dan de vraag kunnen laten beantwoorden door het intoetsen van een "J" of een "N". Bij het stellen van deze vraag zijn er dus maar twee mogelijke acties vanaf het toetsenbord mogelijk: het indrukken van de J-toets of het indrukken van de N-toets. Geen enkele andere toetsaanslag is geldig.

Om te controleren of een geldige toets is ingedrukt, kunnen we nu een Set of Char declareren die de letters "J" en "N" bevat: [’J’,’N’]. De vergelijking "IF TOETSAANSLAG IN [’J’,’N’]" krijgt nu alleen maar de waarde True als er inderdaad een "J" of een "N" is ingedrukt. Alle andere toetsaanslagen krijgen de waarde False. Het aantal verschillende waarden dat in een set kan worden opgeslagen is 255.

Het volgende programma laat zien hoe met een set gewerkt kan worden:

```
PROGRAM TYPE_13;
USES CRT;

VAR
   LETTERS : Set of Char;
   CIJFERS : Set of Byte;
   UITKOMST: Boolean;

BEGIN
   ClrScr;
   LETTERS  := ['A'..'Z','a'..'z'];
   CIJFERS  := [0..5];
   UITKOMST := 'B' IN LETTERS;
   Writeln('De hoofdletter B zit in de set LETTERS ',
            UITKOMST);
   Uitkomst := '?' IN LETTERS;
   Writeln('Het vraagteken zit ook in LETTERS ',
            UITKOMST);
   UITKOMST := 5 IN CIJFERS;
   Writeln('Het cijfer vijf zit in de set CIJFERS ',
            UITKOMST);
   UITKOMST := 9 IN CIJFERS;
   Writeln('Het cijfer 9 zit in de set CIJFERS ',
            UITKOMST);
   Readln
END.
```

Toelichting:

In dit programma is LETTERS gedefinieerd als set voor de hoofdletters "A" tot en met "Z" en voor de kleine letters "a" tot en met "z". De twee puntjes in de declaratie staan voor "tot en met" en de komma scheidt de groepen van elkaar. Als we ook nog de "é" en de "ï" in de set zouden willen opnemen, dan zou er moeten staan:

```
LETTERS := ['A'..'Z','a'..'z','é','ï'];
```

De set wordt gedefinieerd tussen vierkante haken. Met de operator IN kunnen we bepalen of een bepaalde waarde in de set voor komt. De uitkomst van zo'n zoekactie is weer True of False. De Boolean-variabele UITKOMST gebruiken we daarom weer om na te kunnen gaan of een bewering "waar" of "niet waar" is.

Een set kan zowel in de vorm van een variabele als in de vorm van een constante gebruikt worden. De opdracht:

```
   UITKOMST IN LETTERS;
```

heeft hetzelfde effect als:

```
      UITKOMST IN ['A'..'Z','a'..'z'];
```

4.13 Constanten

In het volgende programma worden constanten gedeclareerd die later gebruikt zullen worden om de kleuren in te stellen en om op de juiste plaats een zin te laten verschijnen:

```
PROGRAM TYPE_14;
USES CRT;

CONST
   ACHTERGROND = 4;
   TEKST = 14;
   ZIN = 'Dit is een vaste zin';
   X = 20;
   Y = 10;

BEGIN
   TextBackground(ACHTERGROND);
   TextColor(TEKST);
   ClrScr;
   GotoXY(X,Y);
   Write(ZIN);
   Readln
END.
```

Toelichting:

De waarde van een constante kan tijdens de programma-uitvoering niet gewijzigd worden. Als je geen constanten gebruikt en je wilt later in je programma de achtergrondkleur wijzigen, dan zul je overal in het programma het nummer van de nieuwe kleur moeten invullen bij TextBackground. Als je wel met constanten werkt, hoef je alleen de waarde van de constante ACHTERGROND te wijzigen en het programma opnieuw te compileren. Overal zal nu de gewenste kleur op de bijbehorende waarde gezet worden.

Zoals je ziet, worden constanten in het programma TYPE_14 niet van een type-aanduiding voorzien. Doe je dit wel, dan wordt er gesproken van getypeerde constanten. Een getypeerde constante kan tijdens de uitvoering van het programma wel gewijzigd worden. Eigenlijk is het een variabele die op deze manier van een beginwaarde kan worden voorzien.

Als we het programma TYPE_14 zouden voorzien van getypeerde constanten, zou dat er als volgt uit kunnen zien:

```
PROGRAM TYPE_15;
USES CRT;

CONST
   ACHTERGROND: Byte = 4;
   TEKST: Byte = 14;
   ZIN: String =
        'Dit is een zin die veranderd kan worden.';
   X: Integer = 20;
   Y: Integer = 10;

BEGIN
   TextBackground(ACHTERGROND);
   TextColor(TEKST);
   ClrScr;
   GotoXY(X,Y);
   Write(ZIN);
   Inc(Y);
   ZIN := 'De inhoud van ZIN is nu anders.';
   ZIN := ZIN + ' Y is ook veranderd';
   GotoXY(X,Y);
   Write(ZIN);
   Readln
END.
```

De constanten zijn nu voorzien van een type-aanduiding en van een beginwaarde. De waarde van dit soort constanten kan tijdens de programma-uitvoering wijzigen. Zoals je ziet, gebeurt dat hier ook.

4.14 Opgaven

1. Als een programma gegevens moet inlezen, zullen er variabelen en constanten gedeclareerd moeten worden. Declareer de variabelen en constanten die nodig zijn om de volgende gegevens te lezen:

   a) Voornaam;
   b) Achternaam;
   c) Leeftijd;
   d) Geslacht;
   e) Zakgeld per maand.

   De geldige waarden voor de leeftijd bevinden zich in het bereik 0 tot en met 110. Het geslacht wordt aangeduid met een hoofdletter of een kleine letter: "M", "m" en "V" of "v" zijn mogelijk. De X- en Y-coördinaten worden met behulp van constanten bepaald.

2. Declareer de variabelen en constanten die nodig zijn om voor elke dag van het jaar de gemiddelde temperatuur in graden Celsius te berekenen. Bij de invoer moet ervoor gewaakt worden dat de temperatuur nooit lager dan -20 graden en nooit hoger dan +40 graden kan zijn.

3. Declareer de gegevens die nodig zijn om met een artikelnummer, een artikelnaam en een prijs van een artikel te kunnen werken. Bovendien moet het aantal artikelen aangegeven kunnen worden. De uitkomst van het aantal artikelen vermenigvuldigd met de prijs van het artikel moet op het beeldscherm getoond worden. Het artikelnummer begint altijd met een letter die de categorie aangeeft, gevolgd door een nummer van vier cijfers.

74