11. Dynamische variabelen

11.1 Inleiding

Tot nu toe hebben we op twee manieren variabelen gedeclareerd: als globale variabelen van een programma of een unit en als lokale variabelen binnen een procedure, functie of unit. Deze laatste zijn lokale variabelen, omdat ze uitsluitend gebruikt kunnen worden in de procedure, functie of unit waar ze gedeclareerd staan.

De twee soorten variabelen worden op verschillende manieren in het geheugen opgeslagen. Globale variabelen en constanten worden opgeslagen in het gegevenssegment (datasegment). Het gegevenssegment is een deel van het geheugen waar gegevens opgeslagen worden en kan maximaal 64 Kb groot zijn. Gedurende de hele uitvoering van het programma blijven ze daar staan. Ze kunnen wel een andere waarde krijgen, maar niet verwijderd worden.

Lokale variabelen worden opgeslagen in de stack. De stack is een gedeelte van het geheugen waarvan de afmeting wordt vastgesteld door de programmeur. De programmeur is gebonden aan een maximum van 64 Kb. De vertaling van "stack" is "stapel".

Gegevens worden op de stapel gelegd en daar weer vanaf gehaald op het moment dat dit nodig is. Hierbij wordt het principe gevolgd dat wat het laatste op de stapel gelegd is, er ook weer als eerste afgehaald wordt. Als naar procedure A gesprongen wordt, dan worden de gegevens van A op de stack gezet. Springen we vanuit A naar de procedure B, dan worden de gegevens van de procedure B bovenop de gegevens van A gelegd. Bij het verlaten van de procedure B worden de bijbehorende gegevens van de stack gehaald. We zijn dan weer terug bij procedure A. De gegevens van A liggen nu weer bovenop op de stapel. Verlaten we procedure A, dan worden ook de gegevens van A van de stack gehaald. Hierdoor zijn de waarden in lokale variabelen vluchtig. Na het verlaten van de functie of procedure is alles weg. Het op de stack zetten van gegevens noemt men een "push", het van de stack afhalen heet een "pop".

Het opslaan van variabelen in het gegevenssegment of op de stack heeft nadelen. Het kan zijn dat we met veel grotere gegevensstructuren willen werken dan 64 Kb. Als deze ruimte niet groot genoeg is, loopt ons programma vast. Turbo Pascal biedt ons de mogelijkheid om geheugen te reserveren voor variabelen, met deze variabelen te werken, en als we ze niet meer nodig hebben het geheugen weer vrij te geven. Als we in het verloop van het programma deze variabelen weer nodig hebben, dan claimen we de ruimte opnieuw. Dit soort variabelen worden dynamische variabelen genoemd, in tegenstelling tot statische variabelen in het gegevenssegment. Het werken met dynamische variabelen

vereist een wat andere aanpak dan het werken met statische variabelen, maar is absoluut niet zo moeilijk als vaak gesuggereerd wordt.

11.2 Werken met dynamische variabelen

Zoals gezegd is het geheugen verdeeld in adressen. Zo'n adres bestaat uit twee delen: het segmentdeel en het offsetdeel. Samen vormen deze delen een adres. Om in het geheugen een adres op te slaan, is voor elk deel 2 bytes nodig.

Turbo Pascal levert een gegevenstype dat bestemd is om adressen in op te slaan. Dit gegevenstype wordt "pointer" genoemd. Om een volledig adres op te slaan, zijn 4 bytes nodig. Het adres dat in een pointervariabele staat kan het adres zijn van een veel grotere gegevensstructuur elders in het geheugen.

De plaats waar Turbo Pascal dit soort grotere gegevensstructuren opslaat is de heap. Dit hebben we al in het vorige hoofdstuk kunnen lezen. De heap is de overblijvende geheugenruimte nadat de programmacode, het gegevenssegment, de stack en een eventuele overlay-buffer een plek hebben gekregen. Hoe groter de heap is, des te minder problemen zullen we hebben om gegevens op te bergen.

Het programma POINT_1 laat zien hoe een record van 93 bytes op de heap opgeslagen kan worden en door middel van een pointervariabele van 4 bytes bereikt kan worden. De pointervariabele staat in het gegevenssegment, omdat deze gedeclareerd is als globale variabele. Als zij in een procedure of functie gedeclareerd zou zijn, dan zou bij het verlaten van de functie of procedure ook de variabele verdwenen zijn. Als het record niet meer nodig is, wordt dat stukje geheugen weer vrijgegeven:

PROGRAM POINT_1;

```
USES CRT;

TYPE
   PLedenkaart = ^TLedenkaart;
   TLedenkaart = Record
      Naam     : String[30];
      Adres    : String[30];
      Plaats   : String[20];
      Postcode : String[7];
      Lidnummer: Word;
   END;

VAR
   LEDENKAART: PLedenkaart;
BEGIN
   ClrScr;
   Writeln('Beschikbaar geheugen is: ',MemAvail);
   New(LEDENKAART);
   Writeln('Beschikbaar nadat geheugen geclaimd is: ',
            MemAvail);
   WITH Ledenkaart^ DO
   BEGIN
      Naam      := 'J. Jansen';
      Adres     := 'Dorpsstraat 35';
      Plaats    := 'Ons Dorp';
      Postcode  := '1111 AA';
      Lidnummer :=  100
   END;
   WITH Ledenkaart^ DO
   Writeln('In ledenkaart staat nu: ',Naam,' ',Adres,
           ' ', Plaats,' ',Postcode,' ',Lidnummer);
   Dispose(LEDENKAART);
   Writeln ('Beschikbaar als geheugen vrijgegeven is:',
             MemAvail);
   LEDENKAART := NIL;
   Readln
END.
```

Regels:Toelichting:

[1]3-11Definieer TLedenkaart als een dynamisch record.
[2]12-13 Declareer een dynamische variabele LEDENKAART van het type PLedenkaart.
[3]16 Laat met behulp van MemAvail het beschikbare geheugen zien.
[4]17-19 Declareer geheugenruimte op de heap en laat opnieuw zien hoeveel geheugen er nu beschikbaar is.
[5]20-27Zet gegevens in het record.
[6]28-30 Laat zien wat er op het adres staat waar LEDENKAART naar wijst.
[7]31-33 Geef de geheugenruimte op de heap vrij en laat zien hoeveel geheugen er beschikbaar is.
[8]34 Laat LEDENKAART naar niets wijzen.

Toelichting:

[1]Het eerste wat opvalt bij de definitie van het record is dat er een regel bijgekomen is:

PLedenkaart = ^TLedenkaart

Met deze opdracht geven we aan dat het type PLedenkaart een pointer naar het type TLedenkaart is. Het dakje is het symbool voor een pointer. TLedenkaart is een record met de velden Naam, Adres, enzovoort.

[2]De gedeclareerde variabele LEDENKAART heeft als typering PLedenkaart. De variabele LEDENKAART bevat dus niet zelf de gegevens. Het bevat slechts het adres op de heap waar de velden van het record –dat van het type TLedenkaart is– te vinden zijn.

[3]Turbo Pascal beschikt over een functie MemAvail. Een aanroep van deze functie retourneert het aantal bytes beschikbaar geheugen.

[4]We gebruiken nu de Turbo Pascal-procedure New om ruimte op de heap te declareren. LEDENKAART is een pointer naar TLedenkaart. Als we LEDENKAART als parameter aan New meegeven, dan zoekt New uit hoeveel bytes er op de heap nodig zijn om de velden van het type TLedenkaart te kunnen bevatten. Deze ruimte wordt gereserveerd en het adres van de eerste byte wordt in LEDENKAART gezet. LEDENKAART bevat nu het adres van een record van het type TLedenkaart. MemAvail laat zien dat de hoeveelheid beschikbaar geheugen inderdaad kleiner geworden is.

[5]Hier worden de velden van het record gevuld. Het veld Naam van het record is niet bereikbaar als LEDENKAART.Naam. LEDENKAART is immers geen record, maar een pointer. Het is echter wel een

pointer die getypeerd is naar het type TLedenkaart. Het veld Naam kunnen we nu bereiken met Ledenkaart^.Naam. We gebruiken hier weer het dakje. Ledenkaart^.Naam betekent: het veld Naam van het type TLedenkaart waar LEDENKAART naar wijst.

[6]Zowel bij het vullen als bij het uitlezen van de velden moeten we het dakje gebruiken. Ook kunnen we hier het WITH-statement gebruiken. WITH Ledenkaart^ DO maakt de velden van het type TLedenkaart waar LEDENKAART naar wijst direct toegankelijk.

[7]De Turbo Pascal-procedure Dispose geeft de geheugenruimte weer vrij. Deze procedure doet het tegenovergestelde van de procedure New. New reserveert geheugenruimte en Dispose geeft de gereserveerde ruimte weer vrij. MemAvail rapporteert hier dat het oorspronkelijke geheugen weer beschikbaar is.

[8]Tenslotte geven we de variabele LEDENKAART de waarde NIL. NIL is een pointeradres dat nergens naar wijst. Wanneer Dispose het stukje geheugen heeft vrijgegeven, bevat LEDENKAART nog steeds het adres van de velden van TLedenkaart. Deze locatie is echter vrijgegeven. Het systeem zal de vrijgekomen ruimte mogelijk opnieuw willen gebruiken. Als je na Dispose een pointervariabele uitleest, dan loop je grote kans dat je onbegrijpelijke tekens op je scherm krijgt. Waar je wel voor moet oppassen, is dat je een pointervariabele de waarde NIL geeft nadat je Dispose aangeroept. Doe je dit niet, dan zal het adres van het gereserveerde geheugen verloren gaan. Je kunt dan niet meer bij de gegevens komen, omdat de computer niet meer weet waar ze in het geheugen staan. Omdat het adres verloren is gegaan, kan het gereserveerde geheugen ook niet meer vrijgegeven worden.

NIL kan ook in een vergelijking opgenomen worden, bijvoorbeeld in het geval je alleen iets naar een veld wilt schrijven als er ook inderdaad geheugen gereserveerd is. Bijvoorbeeld:

IF LEDENKAART <> NIL THEN ...

Er bestaat hiervoor zelfs een aparte Turbo Pascal-procedure:

IF Assigned(LEDENKAART) THEN ...

De procedure Assigned retourneert True als er voor LEDENKAART geheugen gedeclareerd is.

Omdat het werken met dynamische variabelen misschien nogal wat vraagtekens oproept, geef ik in het programma POINT_2 een tweede voorbeeld. Dit programma zet een array van 100 bytes op de heap. Na de declaratie van geheugenruimte krijgt elk element van de array een waarde. Tussentijds laat POINT_2 zien hoeveel geheugen er nog beschikbaar is en wat het grootste blok vrije ruimte is:

PROGRAM POINT_2;

```
USES CRT;

TYPE PGetallen = ^TGetallen;
     TGetallen = Array[1..100] of Byte;

VAR
   GETALLEN: PGetallen;
   I       : Byte;

BEGIN
   ClrScr;
   Writeln('Beschikbaar op heap:',MemAvail,
           ' Grootste vrije blok: ',MaxAvail);
   Writeln('Druk op Enter');
   Readln;
   GetMem(GETALLEN,SizeOf(TGetallen));
   Writeln('Beschikbaar na declaratie geheugen:',
            MemAvail,' Grootste vrije blok: ',MaxAvail);
   Writeln('Druk op Enter');
   Readln;
   FillChar(Getallen^,SizeOf(TGetallen),0);
   FOR I := 1 TO 100 DO Getallen^[I] := I;
   FOR I := 1 TO 100 DO Writeln(Getallen^[I]);
   FreeMem(GETALLEN,SizeOf(TGetallen));
   GETALLEN := NIL;
   Writeln('Beschikbaar na vrijgeven geheugen:',
            MemAvail,' Grootste vrije blok: ',MaxAvail);
   Readln
END.
```

Regels: Toelichting:

[1]3-4Definieer een dynamisch type PGetallen als een array van 100 bytes.
[2]5-7Declareer de benodigde variabelen.
[3]10-13 Toon het beschikbare geheugen en het grootste beschikbare blok.
[4]14 Declareer het geheugen voor de dynamische variabele GETALLEN.
15-18Toon het beschikbare geheugen.
[5]19 Zet alle elementen van de array op 0.
[6]20 Vul de array Getallen^ met de waarden 1 tot en met 100.
21 Zet ieder element van Getallen^ op het scherm.
[7]22 Geef de geheugenruimte van de variabele GETALLEN vrij.
[8]23 Laat de variabele GETALLEN naar niets wijzen.
24-26Toon het beschikbare geheugen.

Toelichting:

[1]Het dynamische type TGetallen is een Array [1..100] of Byte. Het spreekt vanzelf dat dit ook een array van records zou kunnen zijn, of welk gegevenstype dan ook. PGetallen is gedefinieerd als pointer naar het type TGetallen.

[2]GETALLEN is een variabele van het type PGetallen. PGetallen is een pointer naar TGetallen.

[3]In het programma POINT_1 hebben we MemAvail al eens gebruikt. MemAvail meldt het totale aantal bytes beschikbaar geheugen. Je kunt je ook voorstellen, dat door het declareren van geheugenruimte voor verschillende variabelen, en het vervolgens weer geheel of gedeeltelijk vrijgeven van die ruimte, het totale beschikbare geheugen zich niet voordoet als één aaneengesloten vrij blok. Tussen de verschillende geheugenlocaties die gebruikt worden voor gegegevensopslag, kan vrije geheugenruimte zitten. De Turbo Pascal-functie MaxAvail geeft de afmeting van het grootste aaneengesloten blok vrije geheugenruimte terug.

[4]In POINT_1 gebruikten we de Turbo Pascal-procedure New voor het declareren van geheugen. De procedure New zoekt uit hoeveel bytes het doorgegeven gegevenstype beslaat en roept vervolgens de Turbo Pascal-procedure GetMem aan. In POINT_2 wordt direct GetMem aangeroepen en wordt het aantal bytes doorgegeven als SizeOf(TGetallen). De SizeOf-functie geeft het aantal bytes terug dat noodzakelijk is om een array van 100 bytes op te slaan. We hadden hier ook

SizeOf(Getallen^) kunnen gebruiken; hiermee wordt hetzelfde aantal bytes teruggeven. SizeOf(GETALLEN), dus zonder dakje, zou maar 4 bytes teruggeven. In dat geval vragen we namelijk om de afmeting in bytes van de pointervariabele GETALLEN. Een pointer gebruikt 4 bytes. Omdat New eerst moet uitzoeken hoeveel bytes er nodig zijn, is GetMem sneller.

[5]Nadat het beschikbare geheugen getoond is, wordt met behulp van de Turbo Pascal- procedure FillChar ieder element van de array op 0 gezet. Na de declaratie van een array bevat elk element van de array de gegevens die toevallig op die plaats in het geheugen stonden. FillChar kan elk element op 0 zetten door bij de aanroep de te wijzigen variabele en de afmeting van de te vullen variabele als VAR-parameter mee te geven. Daarachter wordt vermeld met welk letterteken de bytes gevuld moeten worden. Als het om een numeriek veld gaat, wordt de ASCII-waarde in het veld gezet. Op deze manier kunnen velden dus een beginwaarde van 0 tot en met 255 krijgen.

Let erop dat de variabele als VAR-parameter Getallen^ doorgegeven wordt, en niet als GETALLEN. Als we GETALLEN door zouden geven, zou de pointer met nullen gevuld worden. We zouden dan geen adres meer hebben en de computer zou vreemde dingen kunnen gaan doen.

[6]In een FOR-lus die van 1 tot en met 100 loopt, wordt vervolgens de waarde van I in het corresponderende element van Getallen^ gezet. Een element van de array wordt hier bereikt met Getallen^[I]. Ook hier is weer het dakje geplaatst. Het uitlezen van de elementen gebeurt op dezelfde wijze.

[7]Nu we de array GETALLEN niet meer nodig hebben, kan het geheugen met behulp van FreeMem vrijgegeven worden. We hadden hier ook Dispose kunnen gebruiken, maar deze procedure zou eerst moeten uitzoeken hoeveel bytes er vrijgemaakt moeten worden, en zou vervolgens FreeMem aanroepen. FreeMem moet naast de naam van de pointervariabele ook het aantal bytes doorkrijgen dat vrijgegeven moet worden.

[8]Na het vrijgeven van het geheugen, geven we de pointervariabele GETALLEN de waarde NIL om aan te geven dat zij op dat moment niet naar een geheugenlocatie wijst.

11.3 Het gaat wel eens fout

Vooral in het begin worden er fouten gemaakt, vooral omdat je geneigd bent te vergeten dat dynamische variabelen niet met de gegevens zelf werken, maar met adressen die naar de gegevens wijzen. Ook het declareren en het weer vrijgeven van geheugenruimte leidt regelmatig tot fouten. In het volgende programma staat een fout die snel gemaakt wordt:

```
PROGRAM POINT_3;
USES CRT;
VAR
   NUMMER_1, NUMMER_2: ^Word;
```

```
BEGIN
   ClrScr;
   GetMem(NUMMER_1,SizeOf(Word));
   Nummer_1^ := 100;
   NUMMER_2 := NUMMER_1;
   Writeln('In Nummer_1^ staat:',Nummer_1^);
   Writeln('In Nummer_2^ staat:',Nummer_2^ );
   Writeln('Druk je nu op Enter dan gaat het fout.');
   Readln;
   FreeMem(NUMMER_1,SizeOf(Word));
   FreeMem(NUMMER_2,SizeOf(Word))
END.
```

In POINT_3 worden twee variabelen gedeclareerd als pointer naar het type Word. Voor de variabele NUMMER_1 wordt geheugen gedeclareerd. Op de plaats waar de variabele NUMMER_1 naar wijst, wordt het getal 100 gezet. Als we daarna opgeven:

NUMMER_2 := NUMMER_1;

dan wordt het adres dat in NUMMER_1 staat ook aan NUMMER_2 gegeven. In dat geval hebben we twee pointervariabelen die naar hetzelfde adres wijzen. Dat blijkt wel als we met een Writeln-opdracht de inhoud van Nummer_1^ en Nummer_2^ op het beeldscherm laten tonen. Ze geven beide de waarde 100. Als daarna de ruimte van NUMMER_1 vrijgegeven wordt, gaat nog steeds alles goed. Pas als we proberen om ook de geheugenruimte van NUMMER_2 vrij te geven, worden we gestraft met foutmelding 204: "invalid pointer operation". Natuurlijk heeft het systeem gelijk. We hebben maar één keer 2 bytes ruimte gedeclareerd. Dat was toen we NUMMER_1 naar GetMem stuurden. Daarna hebben we het adres van NUMMER_1 in NUMMER_2 gezet. Met andere woorden: NUMMER_1 en NUMMER_2 wezen naar hetzelfde adres op de heap. Als je dan vervolgens twee keer ruimte probeert vrij te geven die maar eenmaal gedeclareerd is, gaat het fout.

11.4 Opgaven

11.1 a. Declareer een dynamische variabele ZIN van het type String.

b. Declareer een dynamisch record met de velden Naam, Adres en Woonplaats.

c. Declareer een dynamische Array [1..1000] of Char.

d. Declareer een dynamische Array [1..100] of Pointer.

e. Declareer een variabele Pointers als Array [1..100] of Pointer.

11.2Maak een programma dat dynamisch een Array [1..20] of Char declareert. Het geheugen voor deze array moet op de heap gereserveerd worden. De array moet volgezet worden met de letter ’A’. De inhoud van de array moet naar het scherm gestuurd worden. Daarna moet het geheugen vrijgegeven worden en moet de pointervariabele naar niets wijzen.

199

199

207

207