Turbo Pascal deel 7

Erik van Bilsen

MSX Club Magazine 40

Scanned, ocr'ed and converted to PDf by HansO, 2001

In deze aflevering van deze serie zal ik proberen een groot gebrek van Turbo Pascal weg te nemen, namelijk het gemis aan Random Access Files.

Random Access Files

Zoals ik in aflevering 5 heb belicht biedt Turbo Pascal de prachtige mogelijkheid om records te definiëren. Een groot nadeel is echter dat Turbo Pascal geen Random Access Files kent, zodat het maken van grote databases weinig zin heeft. Wat is nu precies een Random Access File (RAF). Een RAF biedt de mogelijk om gegevens, records, rechtstreeks midden in een bestand te lezen of te beschrijven, dit in tegenstelling tot sequentiële files die achtereenvolgens gelezen of beschreven moeten worden. Een voorbeeld: een sequentieel bestand bevat 1000 records. Wil je nu het 900ste record lezen, dan moeten eerst de 899 voorgaande records worden ingelezen, hetgeen erg veel tijd kost. Dit is de mogelijkheid die Turbo Pascal biedt. Een oplossing hiervoor is een Random Access File, vertaald een bestand waartoe je willekeurig toegang hebt. Met een RAF is het mogelijk om rechtstreeks het 900ste record te lezen of te beschrijven. RAF's zijn wel in BASIC bekend en zijn bestanden die worden gedefinieerd met FIELD. Een voorwaarde voor een RAF is natuurlijk wel dat elk record even groot is, zodat kan worden berekend waar het 900ste record zich in het bestand bevindt. Daarom moet bij het openen van een RAF in BASIC de lengte worden opgegeven m.b.v. LEN=.

R_FILES.LIB

Voor degenen die niet zozeer geïnteresseerd zijn in het HOE en WAT van deze procedures, volgt hier eerst uitleg over hoe je de procedures moet gebruiken. Aan het begin van je programma moet je de constante MaxFiles definiëren, die het maximaal aantal gelijk geopende RAF's bevat (gelijk met MAXFILES in BASIC). Bijvoorbeeld:

```
CONST
  MaxFiles = 3 ;
```

Vervolgens moetje de include-fi-les MSXBIOS.LIB en R_FILES.LIB in je programmatekst opnemen:

```
{$I   MSXBIOS.LIB}
{$I   R_FILES.LIB}
```

Maak je in je programma verder geen gebruik van het MSX-Bios of van procedures die er gebruik van maken (zoals de grafische procedures in de eerste afleveringen), dan kun je de regel {$I MSXBIOS .LIB) vervangen door:

VAR
  regA, regDE, regHL: INTEGER;

```
PROGRAM R_Test;

CONST
   MaxFiles = 3;

VAR
   Klant: RECORD
             Nummer: INTEGER;
             Naam: STRING[10]
          END;
   regA, regDE, regHL: INTEGER;

{$I R_FILES.LIB}

BEGIN
   R_Open('KLANTEN.DAT',2,SizeOf(Klant));
   R_Field(2,Klant);
   Klant.Nummer:=100; Klant.Naam:='Anton'; R_Put(2,1);
   Klant.Nummer:=200; Klant.Naam:='Bart'; R_Put(2,2);
   R_Get(2,1);
   Writeln(Klant.Nummer,': ',Klant.Naam);
   R_Close(2)
END
```

Een voorbeeld

Een voorbeeld van het gebruik van RAF's is te vinden in het programma RJTest, bovenaan deze pagina. In dit voorbeeld gaan we een klantenbestand aanmaken. Daarbij bestaat elk klantenrecord uit een nummer (integer) en een naam (string). Achtereenvolgens zal ik de gebruikte procedures aan bod laten komen. Daarbij begint de naam van elke RAF-procedure met een R en een underscore (R_). Ten eerste wordt een klantenbestand geopend m.b.v. de procedure R_Open. De parameters zijn de naam en het nummer van het bestand en de recordlengte, bijvoorbeeld:

R_0pen('TEST.DAT',1,6);

Dit komt overeen met in BASIC:

OPEN "TEST.DAT" AS#1 LEN=6

Zoals je ziet wordt in het voorbeeld gebruik gemaakt van de standaardprocedure SizeOf.

SizeOf

De standaardprocedure SizeOf wordt gebruikt om de lengte van een gegeven in bytes te bepalen. Zo is de lengte van een INTEGER 2 bytes en de lengte van een REAL 6 bytes. De procedure SizeOf kan worden toegepast op alle soorten gegevenstypes, dus ook op array's, records en enumeratietypen, bijvoorbeeld:

```
VAR
  TestArray: ARRAY [1. .10] OF INTEGER;

BEGIN
  WriteLn  (SizeOf(TestArray))
END;
```

In het voorbeeld RJTest wordt de procedure SizeOf gebruikt voor de lengte van een klantenrecord. Dit record bestaat uit een string van 10 karakters en een integer. De lengte van dit record is dus 13 bytes (10 bytes voor de karakters, l byte voor de werkelijke lengte van de string en 2 bytes voor de integer). Het RAF KLANTEN.DAT bestaat dus uit blokken van 13 bytes.
Vervolgens wordt met de procedure R_Field aangegeven uit welke gegevens het RAF bestaat, in dit geval dus uit een klantenrecord. Met behulp van R_Put kan dit klantenrecord naar het bestand worden geschreven. Daarbij wordt het bestandsnummer en het re-cordnummer opgegeven. R_Put (2,3) komt dus overeen met in BASIC: PUT #2,3. Op dezelfde manier kunnen met R_Get records uit het bestand worden gelezen. Het is belangrijk om aan het eind van het programma het bestand af te sluiten met R_Qose omdat er anders gegevens verloren kunnen gaan.

Foutmeldingen

Het werken met bestanden kan fouten opleveren. Zo kan bijvoorbeeld de diskette vol zijn of beveiligd zijn tegen schrijven. Ik heb hiermee rekening gehouden door gebruik te maken van een enume-ratie-variabele met de naam R_Er-ror. Deze variabele kan de volgende waarden hebben: NoError, DiskFull, EOF, en FileNotOpen (let op gebruik van hoofd- en kleine letters). Normaalgesproken bevat deze variabele na afloop van een RAF-procedure de waarde NoError. Als bij het openen of beschrijven van een bestand de diskette vol is heeft deze variabele de waarde DiskFull. De waarde EOF komt voor als een recordnummer wordt gelezen dat niet bestaat, en de waarde FileNotOpen als een bestand wordt gesloten dat niet geopend is. Op de volgende manier kan van de variabele R_Error gebruik worden gemaakt

```
IF  R_Error=DiskFull THEN WriteLn('De diskette   is  vol');
```

Met behulp van de RAF-proce-dures kan snel van het ene naar het andere record worden gesprongen. Deze proce-
dures zijn dus ideaal voor het opzetten van een database. Met de combinatie van de recordvormen van Turbo Pascal en de Random Access Files-routines uit deze aflevering

heb je dus een krachtig hulpmiddel om op een eenvoudige manier professionele databases op te zetten.

BDOS en FCB

De routines voor het behandelen van Random Access Files maken gebruik van het BDOS, het Basic Disk Operating System. Het BDOS, aanwezig in het disk-ROM van de computer, bevat routines voor de behandeling van diskdrive en bestanden. Ik zal niet uitvoerig op BDOS ingaan, wellicht wordt hierover meer verteld in de machinetaalcursus ? De Bdos-routine in Turbo Pascal kan op dezelfde manier worden aangeroepen als de MSX-Bios routines in het bestand MSXBIOS.LIB. De registers DE en HL (regDE en regHL) worden gevuld met een bepaalde waarde en vervolgens wordt de routine Bdos aangeroepen met als parameter het nummer van de functie, bijvoorbeeld Bdos($lA);. In de constantendeclaratie van het bestand RJ1LES.LIB staan enkele functie-nummers met hun namen. De Bdos-routines maken vaak gebruik van het zogenaamde File Control Block (FCB). Dit is een stukje geheugen van 37 bytes dat begint op adres $005C (onder DOS) en informatie over een bestand bevat, zoals de naam, de grootte, de datum en tijd van creatie enzovoorts. Ook hierop zal ik niet diep ingaan (hoewel een artikel over het omgaan met de diskdrive in machinetaal voor enkele misschien erg interessant is. Als dat zo is, laat het dan even weten!). Zoals je ziet in de variabelendeclaratie is het FCB een array van 37 bytes, beginnend op het absolute adres $5C. Verder worden er hulp-FCB's aangemaakt voor het maximum aantal bestanden (de variabele RJFiles).
Na de procedure Bdos word R_Open gedeclareerd, met als parameters de bestandsnaam, bestandsnummer en recordlengte. Ten eerste wordt de bestandsnaam overgezet in een hulp-FCB (bytenummers 1-11). Daarna wordt deze hulp-FCB gekopieerd naar hetFCB.

Kopiëren van arrays

Het kopiëren van gelijksoortige arrays kan met l commando. Als bijvoorbeeld de variabelen A en B beide array's zijn van 10 integere, dan kan de array A worden gekopieerd naar array B met behulp van B: =A;. De volgende constructie, zoals in BASIC is dus niet nodig:

```
FOR Teller:=1   TO   10   DO B[Teller]:=A[Teller];
```

Dit is eigenlijk heel logisch, want alle andere gegevenstypen zoals re-als en strings kunnen ook met één commando worden gekopieerd, en een string is niets meer dan een array van karakters.
Terug naar de routine. Met behulp van Bdos-routine $OF (OpenFik), wordt getracht het bestand zoals aangegeven in het FCB te openen. Lukt dit niet, bijvoorbeeld doordat het bestand nog niet bestaat, dan bevat regA de waarde 255. In dat geval wordt het bestand op schijf gecreëerd. Wil ook dat niet lukken, dan krijgt de variabele R_Error de waarde DiskFull. Tot slot wordt byte 14 en 15 van het FCB gevuld met de recordlengte.

Ongetypeerde VAR-parameters

Speciale aandacht vraagt de procedure R_Field. De aanhef van die procedure is als volgt
PROCEDURE R_Field (FileNr: BYTE; VAR Gegevens);
Als je goed kijkt moet iets vreemds opvallen. Van de parameter Gegevens is namelijk geen gegevenstype opgegeven. Het is dus onbekend of dit een integer, real, array, enumeratietype of record is. Turbo Pascal biedt de mogelijkheid om in de procedure-aanroep het gegevenstype van de parameter in het midden te laten. Zo'n parameter heet dan een ongetypeerde VAR-parameter. Het voordeel hiervan is dat met één procedure verschillende gegevenstypen kunnen worden verwerkt. Dit is bijvoorbeeld handig voor een variant van het SW AP-commando in BASIC om twee variabelen van elkaar te wisselen, ongeacht of het een integer of een real of wat dan ook is. Als je het onderscheid tussen gewone en VAR-parameters goed begrepen hebt, moet duidelijk zijn waarom voor een ongetypeerde parameter altijd het woord VAR moet. Maar wat doet de procedure R_Field nu. In de array Address wordt het geheugenadres van de ongetypeerde variabele gezet, met behulp van de standaardprocedure Addr. Dit adres wordt door de procedure^ R_Get gebruikt om alle gegevens die van schijf worden gelezen vanaf dat adres in het geheugen weg te zetten (in het voorbeeld RJTest komen de ingelezen gegevens dus in het klantenrecord terecht). Hetzelfde geldt voor R_Put. In deze twee procedures wordt het record-nummer geplaatst in bytenummers $21 en $22 (33 en 34) van het FCB en wordt de variabele regHL gevuld met het aantal records, in dit geval 1. Vervolgens wordt de Bdos-routine voor het lezen of schrijven van een random record aangeroepen en wordt de variabele R_Error bijgewerkt.

```
{ ***********************************
  *           R_FILES.LIB           *
  *                                 *
  * Procedures voor Random Access   *
  * Files in Turbo Pascal           *
  *                                 *
  *       TRIPLE SOFT (C) 1992      *
  *       door Erik van Bilsen      *
  *                                 *
  * Bijlage bij cursus Turbo Pascal *
  * MSX Club Magazine 40            *
  *********************************** }

CONST
   SelectDisk = $0E;
   OpenFile = $0F;
   CloseFile = $10;
   CreateFile = $16;
   SetTransferAddress = $1A;
   RandomBlockWrite = $26;
   RandomRecordRead = $27;

TYPE
   FCB_Array = ARRAY [0..36] OF BYTE;
   Str12 = STRING[12];

VAR
   FCB: FCB_Array ABSOLUTE $5c;
   R_Files: ARRAY [1..MaxFiles] OF FCB_Array;
   Address: ARRAY [1..MaxFiles] OF INTEGER;
   R_Error: (NoError,EOF,DiskFull,FileNotOpen);

PROCEDURE Bdos (Entry: INTEGER);
BEGIN
   InLine ($ed/$4b/Entry/    { ld bc,(Entry) }
           $ed/$5b/regDE/    { ld de,(regDE) }
           $2a/regHL/        { ld hl,(regHL) }
           $cd/$05/$00/      { call $0005    }
           $32/regA/         { ld (regA),a   }
           $ed/$53/regDE/    { ld (regDE),de }
           $22/regHL/        { ld (regHL),hl }
           $fb)              { ei            }
END;

PROCEDURE R_Open (FileName: Str12; FileNr,Len:BYTE);
VAR Teller,Index: BYTE;
BEGIN
   regA:=0; Index:=1;
   FOR Teller:=0 TO 36 DO R_Files[FileNr,Teller]:=0;
   FOR Teller:=1 TO 11 DO R_Files[FileNr,Teller]:=32;
   FOR Teller:=1 TO Length(FileName) DO
   IF FileName[Teller]='.' THEN Index:=9 ELSE
      BEGIN
         R_Files[FileNr,Index]:=Ord(FileName[Teller]);
         Index:=Index+1
```

```
      END;
   FCB:=R_Files[FileNr];
   regDE:=$5C; Bdos(OpenFile);
   R_Error:=NoError;
   IF regA<>0 THEN
      BEGIN
         regDE:=$5C; Bdos(CreateFile);
         IF regA<>0 THEN R_Error:=DiskFull
      END;
   FCB[$0e]:=Len; FCB[$0f]:=0;
   R_Files[FileNr]:=FCB
END;

PROCEDURE R_Close (FileNr: BYTE);
BEGIN
   FCB:=R_Files[FileNr];
   regDE:=$5C; Bdos(CloseFile);
   R_Error:=NoError;
   IF regA<>0 THEN R_Error:=FileNotOpen
END;

PROCEDURE R_Field (FileNr: BYTE; VAR Gegevens);
BEGIN
   Address[FileNr]:=Addr(Gegevens)
END;

PROCEDURE R_Get (FileNr: BYTE; RecordNr: INTEGER);
BEGIN
   regDE:=Address[FileNr]; Bdos(SetTransferAddress);
   R_Files[FileNr,$21]:=Lo(RecordNr);
   R_Files[FileNr,$22]:=Hi(RecordNr);
   FCB:=R_Files[FileNr];
   regHL:=1; regDE:=$5C; Bdos(RandomRecordRead);
   R_Error:=NoError;
   IF regA<>0 THEN R_Error:=EOF;
   R_Files[FileNr]:=FCB
END;

PROCEDURE R_Put (FileNr: BYTE; RecordNr: INTEGER);
BEGIN
   regDE:=Address[FileNr]; Bdos(SetTransferAddress);
   R_Files[FileNr,$21]:=Lo(RecordNr);
   R_Files[FileNr,$22]:=Hi(RecordNr);
   FCB:=R_Files[FileNr];
   regHL:=1; regDE:=$5C; Bdos(RandomBlockWrite);
   R_Error:=NoError;
   IF regA<>0 THEN R_Error:=DiskFull;
   R_Files[FileNr]:=FCB
END;
```